

Microsoft Excel 95-365



ISOZ Vincent

Notes de cours V15.0 (2024-01-16)

{oUUID 1.828}

Please consider the environment - do you really need to print this document!?

TABLE DES MATIÈRES

1. ABSTRACT

Remarques:

- R1. Ce support est quasi-inutile pour un non initié si celui-ci n'a pas suivi le cours dispensé y relatif et qu'il n'a pas complété ce dernier par de nombreuses annotations.

- R2. Ce support constitue un "super condensé" de ce qui est vu pendant le cours et qui pourrait facilement tenir sur plusieurs milliers de pages.

- R3. Nous avons exprès introduit des erreurs et des incohérences dans le document afin d'exciter l'esprit critique des apprenants

- R4. La formation de deux 2 jours "V.B.A. MS Excel initiation" et 2 jours "V.B.A. MS Excel avancé" est objectivement irréaliste pour avoir une introduction rigoureuse à ce langage de programmation. Dans un cas idéal (scolaire, tel qu'université ou école d'ingénieurs) la durée cumulée est de 5 jours d'initiation et 5 jours de cours avancés en considérant que l'algorithmique est déjà connue (et encore…).

- R5. Il peut arriver à tout moment que Microsoft décide suite à des mises à jour (Windows Update) que certaines commandes ou ActiveX soient désactivés. C'est ainsi et vous ne pourrez rien y faire. Il faudra réadapter votre code….

Donc:

CE DOCUMENT EST PRÉVU POUR ÊTRE ACCOMPAGNÉ D'EXPLICATIONS ORALES ET DE DOCUMENTS ANNEXES. IL EST DONC TRÈS LOIN D'ÊTRE COMPLET.

CECI IMPLIQUE QUE:

SI VOUS N'AVEZ PAS ASSISTÉ AU COURS, LA LECTURE DES PAGES SEULES PEUT VOUS AMENER À FAIRE DES CONTRESENS IMPORTANTS ET **DANGEREUX**

R6. Nous tenons à nous excuser du fait que ce document mélange les captures d'écran de pas mal de versions de MS Excel (97 à 2003) et en plus de différentes langues. Effectivement, notre métier de consultant en formation professionnelle nous oblige constamment à changer d'ordinateurs et ainsi les contenus des documents que nous rédigeons. Nous espérons que le lecteur comprendra en attendant une uniformisation.

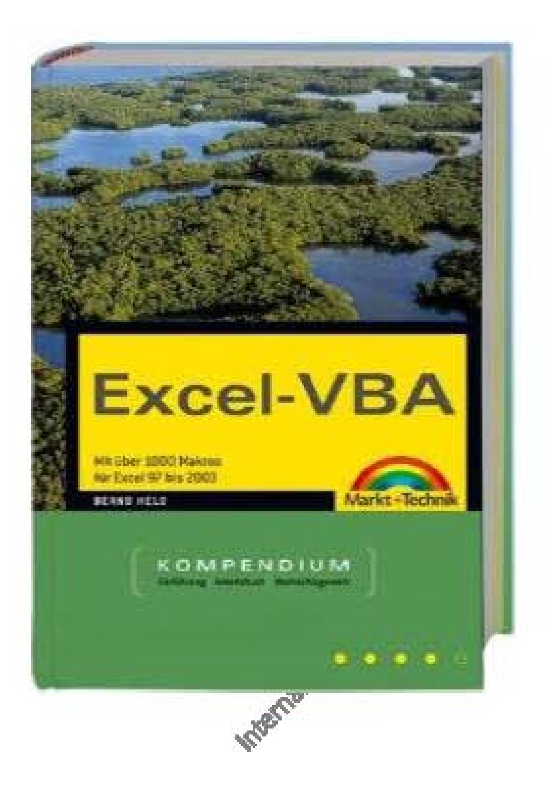
Remerciements:

Je tiens à remercier particulièrement Olivier WEBER, Formateur et Consultant Senior et grand spécialiste V.B.A. (par ailleurs…), pour son aide et ses précieux conseils dans l'élaboration de ce document. Un grand merci aussi pour ses corrections à tous niveaux et d'avoir pris sur son temps (précieux).

Des remerciements aussi à Fabrice FOURNIER pour les nombreux compléments.

2. RÉFÉRENCES BIBLIOGRAPHIQUES

S'il fallait n'en citer qu'un (bon il est en allemand mais il excelle dans son domaine) *Excel-V.B.A. Kompendium* (1000 macros pour Excel 97 à 2003), ISBN-13: 978-3827265777, 928 pages, Bernd Held, Editions Markt+Technik:



3. VOTRE AVIS NOUS INTERESSE!

En tant que lecteur de ce document, vous êtes le critique et le commentateur le plus important. Votre opinion compte et il est très intéressant de savoir ce qui est bien, ce qui peut être mieux et les sujets que vous souhaiteriez voir être traités.

Vous pouvez nous envoyer un courriel pour partager ce que vous avez aimé ou détesté dans le présent document afin d'en assurer une amélioration continue.

Notez que malheureusement, nous ne pouvons répondre gratuitement à des questions techniques d'ingénierie ou de problématique d'entreprise par e-mail pour des raisons professionnelles évidentes.

E-mail: isozv@outlook.com

4. INTRODUCTION

Ce cours s'adresse à des personnes n'ayant peu ou pas d'expérience de la programmation et désireuses de développer par la suite des documents interactifs en intégrant dans les applications MS Office des macros automatiques et du code V.B.A.

Seront particulièrement concernées par ce cours:

- Les personnes avec un esprit logique et mathématique qui désirent avoir un premier contact avec le monde de la programmation.

- Les personnes ayant à automatiser des tâches sous MS Excel.

- Les personnes ayant créé quelques macros et qui veulent pouvoir en comprendre le contenu et en assimiler les subtilités et voir les possibilités.

De bonnes connaissances d'un ou plusieurs des outils de la suite MS Office est souhaitable. Une approche rigoureuse de l'informatique est essentielle (génie logiciel, algorithmique, analyse numérique…).

Pour plus d'informations sur l'algorithmique, l'histoire des langages de programmation ou la norme syntaxique habituelles de codage, veuillez-vous référer aux documents téléchargeables sur Internet ou demander à votre formateur.

Le V.B.A. est un langage de programmation (non réellement orienté objet) utilisé par et pour les applications MS Office listées ci-dessous:

- MS Word
- MS Excel
- MS Access (voir le cours correspondant téléchargeable sur Sciences.ch)
- MS Visio
- MS Publisher
- MS Project (voir le cours correspondant téléchargeable sur Sciences.ch)
- MS Outlook (à partir de la version 2002… du moins facilement)
- MS FrontPage
- MS PowerPoint

Ce langage est simple d'utilisation et n'a absolument aucun commun rapport avec le langage Visual Basic .Net (nous considérons le langage Visual Basic 6 comme mort dans ce cours). La plus grosse différence étant que le V.B.A. ne permet pas de faire ce que nous nommons des applications en "Standalone". Nous utilisons normalement les macros ou le V.B.A. dès que les outils WYSIWYG des logiciels de la suite MS Office ne satisfont plus nos besoins.

Enfin, rappelons qu'avant d'écrire un programme quelconque, la première des choses à faire est d'éteindre son ordinateur et de réfléchir. On peut notamment se poser les questions suivantes:

- Quel est l'objectif de mon code?

- N'est-il pas plus rapide de réaliser cet objectif manuellement? (calul du ROI)

- Cet objectif a-t-il réellement un intérêt?
- Ce programme n'existe-il pas déjà sous une autre forme?
- Ce programme est-il réalisable?
- La réalisation de ce programme n'est-elle pas trop coûteuse?

Bien évidemment, il existe de nombreux cas où vous pourrez écrire un programme sans vous poser toutes ces questions. Ainsi, quand vous voudrez rédiger un code très simple pour automatiser une tâche précise qui n'est pas complexe, vous pourrez foncer bille en tête. En revanche, dès que le projet de code devient un peu plus amitieux, il vaut vraiment mieux se poser des questions avant de commencer à écrire du code.

Un ordinateur est un outil incomparable entre les mains de celui qui sait. Sous les doigts du Crétin, c'est un revolver manip par un aveugle au milieu de la foule Chester Himes

4.1 Objectifs

A la fin de cette formation, les participants sauront parfaitement utiliser les macros automatiques et macro complémentaires et connaîtront les notions et structures standards de la programmation VBA, telles que les variables, les boucles, les conditions et les fonctions.

Ils sauront ce que sont la programmation objet et la programmation évènementielle et auront réalisés quelques exercices utilisant ces notions.

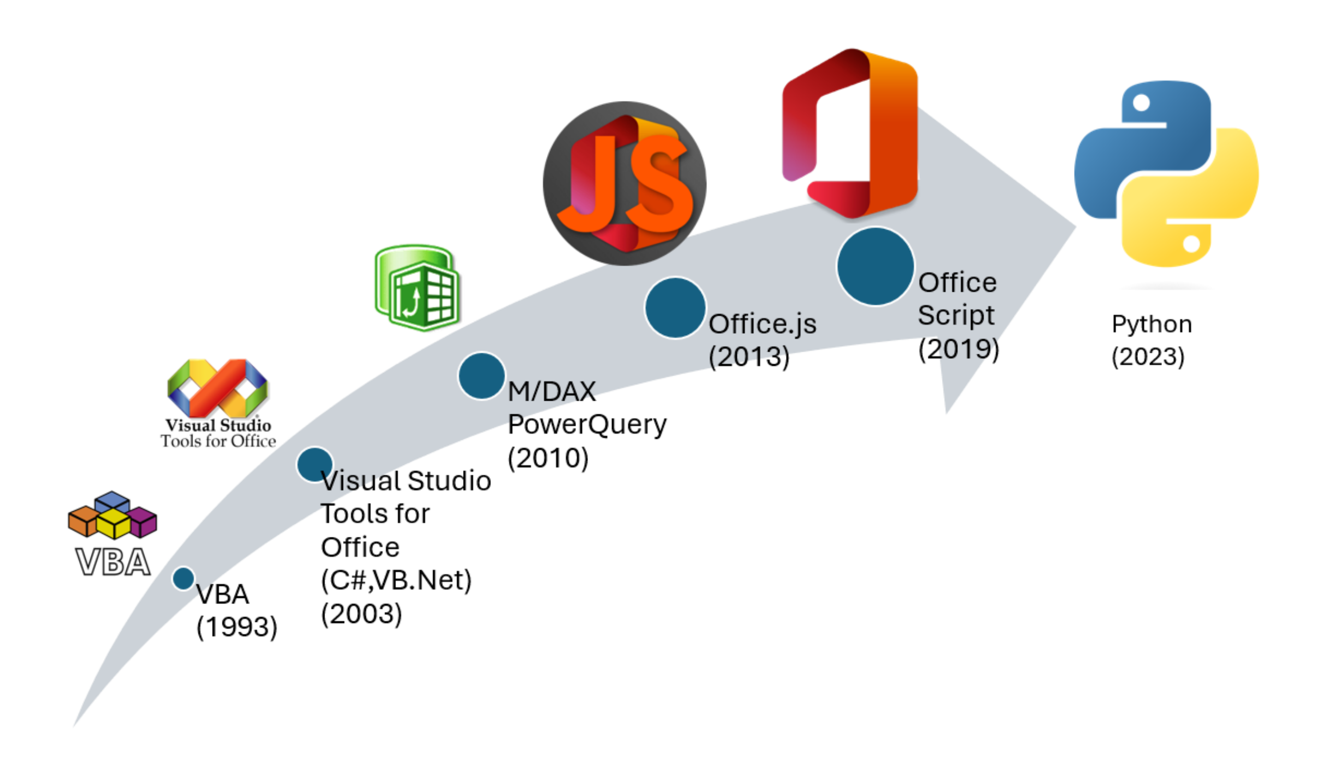
Ils auront travaillé avec des formulaires et manipulé des composants utilisateurs simples comme des boutons et des champs texte.

Remarque: On sait que le nombre de mots d'une langue est limité. Le vocabulaire d'un enfant de 10 ans tourne autour de 5'000 mots, celui d'un adulte cultivé de 10'000-15'000, et les dictionnaires en plusieurs volumes peuvent monter de 130 000 à 200 000. Le V.B.A. d'après des estimations contiendrait environ 800'000 mots… (à vérifier quand même!). J'ai contrôlé qu'avec les options par défaut (donc avec le minimum on était a environ entre 2'000 et 2'500 mots)... donc avec toutes les références possibles activées on ne doit pas être loin de la vérité

4.2 Historique

En programmation, **BASIC** est un acronyme pour **Beginner's All-purpose Symbolic Instruction Code** qui désigne une famille de langages de programmations de haut niveau.

Le BASIC a été conçu à la base en 1963 par le mathématicien John George Kemeny (1926-1993) et le mathématicien Thomas Eugene Kurtz (1928-) au Dartmouth College pour permettre aux étudiants qui ne travaillaient pas dans des filières scientifiques d'utiliser les ordinateurs et apprendre les techniques de programmation. En effet, à l'époque, l'utilisation des ordinateurs nécessitait l'emploi d'un langage de programmation réputé réservé aux seuls les spécialistes, en général un langage d'assemblage ou Fortran.



L'acronyme BASIC est lié au titre d'un article de Thomas Kurtz qui n'a pas été publié et n'a aucun rapport avec les séries intitulées « Anglais basic » de C. K. Ogden. Les concepteurs du langage souhaitaient qu'il soit du domaine public, ce qui favorisa sa diffusion.

Le BASIC est indissociable de l'apparition, dans les années 1980, de la micro-informatique grand public. En effet, la plupart des micro-ordinateurs vendus durant cette période étaient fournis avec un Interprète BASIC, et quelques calculatrices programmables en furent même dotées.

Ce n'était jamais l'intention des créateurs du langage qu'il s'agit d'un langage professionnel. Pourtant il s'est propagé rapidement et est disponible dans les centaines de dialectes sur de nombreux types d'ordinateurs. La BASIC a évolué et s'est amélioré au cours des années. À l'origine, c'était un langage interprété (chaque ligne était interprétée avant son exécution… ce qui est le cas du V.B.A. par exemple…) qui impliquait une exécution lente. La plupart des dialectes modernes de BASIC permettent au code d'être compilé. Par conséquent, l'exécution est beaucoup plus rapidee et la portabilité des programmes améliorée.

Les huit principes de conception du BASIC étaient:

1. Être facile d'utilisation pour les débutant(e)s (Beginner)
2. Être un langage généraliste (All-purpose)
3. Autoriser l'ajout de fonctionnalités pour les expert(e)s (tout en gardant le langage simple pour les débutant(e)s)
4. Être interactif
5. Fournir des messages d'erreur clairs et conviviaux
6. Avoir un délai de réaction faible pour les petits programmes
7. Ne pas nécessiter la compréhension du matériel de l'ordinateur
8. Isoler (shield) l'utilisateur du système d'exploitation

Le BASIC a gagné sa respectabilité en 1991 lorsque Microsoft a lancé VISUAL BASIC pour MS Windows. Ce produit a été très populaire parmi les développeurs d'applications autonomes. Si V.B.A. ressemble peu à ces langages, le BASIC reste la base sur laquelle V.B.A. a été élaboré.

EXCEL 5 a été la première application sur le marché à proposer V.B.A. et il est maintenant inclus dans presque toutes les applications de la suite bureautique depuis MS Office 97 et même chez d'autres fournisseurs. Par conséquent, si vous maîtrisez l'utilisation de VBA, vous pouvez écrire des macros avec toutes sortes d'applications (Microsoft et autres).

Il est important de noter l'information suivante (capture d'écran du site web de Microsoft):

Visual Basic for Applications

Discontinuation of the VBA Licensing Program

Since June 1996, when we first announced the Microsoft® Visual Basic® for Applications (VBA) licensing program, we have been offering VBA for licensing to Independent Software Vendors and others who wished to integrate VBA into their own applications. As previously announced, Microsoft does not expect to make significant enhancements to VBA. This does not impact the current support commitments for VBA in any way, and of course, it does not impact any license arrangements that are in force. In particular, this does not impact VBA in Microsoft Office products.

Microsoft is investing its application programmability resources in Microsoft® Visual Studio® Tools for Applications (VSTA) and its companion set of tools, Microsoft® Visual Studio® Tools for Office (VSTO). We encourage you to consider VSTA for new applications that require application programmability technology. Summit Software is Microsoft's vendor for VSTA licensing.

As of July 1, 2007, Microsoft will no longer offer VBA distribution licenses to new customers. Existing VBA customers can still purchase additional VBA licenses from Summit Software and Microsoft for existing solutions.



4.3 Avertissements

Les avertissements ci-dessous, qui s’appliquent aussi aux autres logiciels de la suite Microsoft Office s’adressent particulièrement aux débutants découvrant les Macros/VBA afin de prendre les meilleures décisions pour une gouvernance à long terme de leurs macros :

- Une macro (ou code VBA) créée pour une version donnée de Microsoft Excel a de fortes chances de ne plus fonctionner assez rapidement (3-6 ans) dans les versions ultérieures de ce logiciel

- Les macro (ou codes VBA) ne peuvent quasiement rien faire au niveau du contrôle de Power Query, Power Pivot, Power Map et Power View.

- Si vous voulez éviter des complications "subtiles", ne nommez jamais vos feuilles Microsoft Excel avec des accents et espaces.

- Faites si possible un copie de votre fichier avant d'y modifier vos Macros ou codes VBA (excepté si vous travaillez dans une GED faisant du versionning automatique).

4.4 Liens Internet

http://www.google.com

http://www.youtube.com

http://support.microsoft.com/newsgroups

http://www.excel-vba.com

http://www.info-3000.com

http://www.vbaexpress.com

http://jacxl.free.fr

http://www.excel-downloads.com

http://www.spreadsheetworld.com

http://www.wrox.com

http://www.vbfrance.com

http://www.mrexcel.com

Site V.B.A. officiel de Microsoft:
http://ww.iheartmacros.com

Quelques propriétés, méthodes et classes nouvelles et changées dans Excel 2010 x86/x64:
http://msdn.microsoft.com/en-us/library/ff846371.aspx

Liste de quelques changements du V.B.A. entre Excel 2003 et Excel 2010:
http://msdn.microsoft.com/library/ee836187%28office.14%29.aspx#XL14DevRef_ChangesSince2003

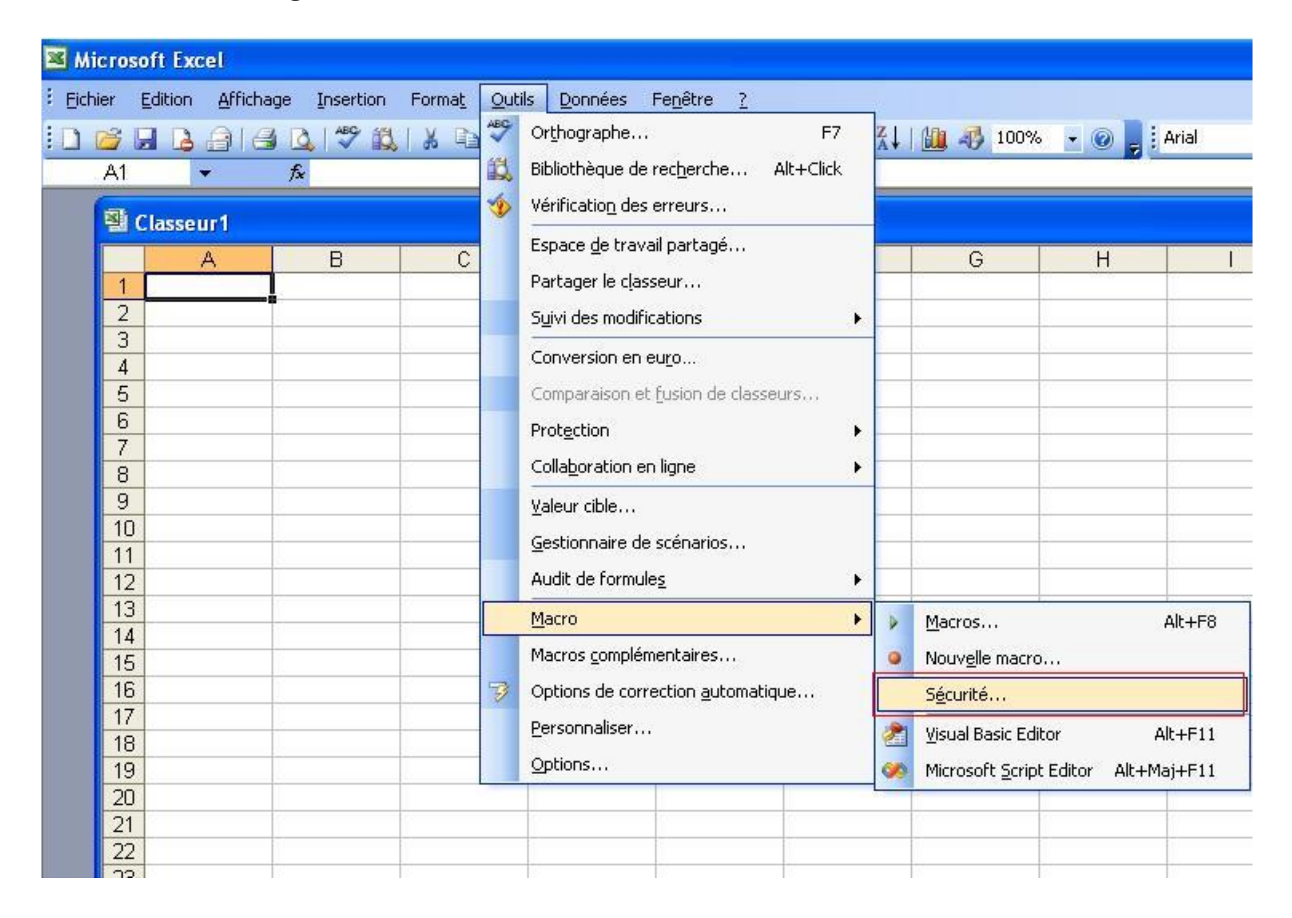
Office Code Compatibility Inspector (OCCI):
http://www.microsoft.com/en-us/download/details.aspx?id=15001

Add-in pour améliorer l'éditeur VBA:
https://www.mztools.com/

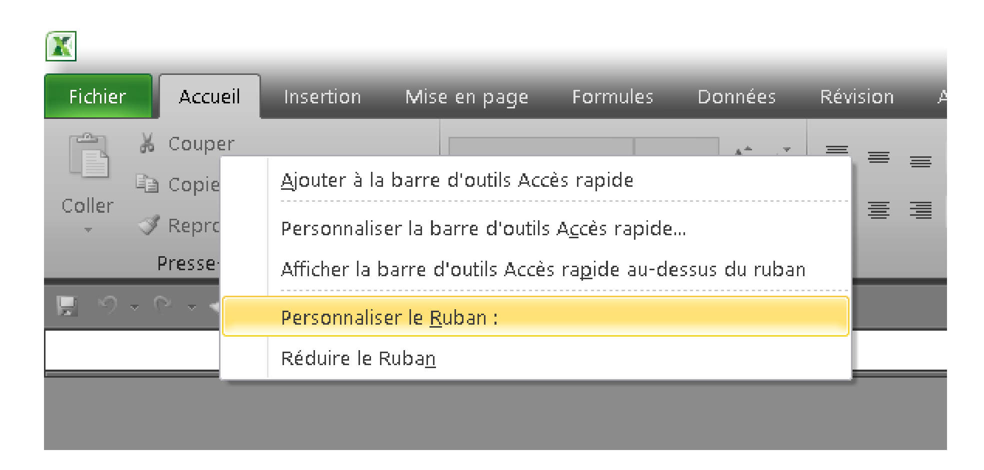
Autre add-in pour améliorer l'éditeur VBA et aussi accélérer consdiérablement l'écritue de certains codes:
https://www.automateexcel.com/

5. OUTILS MACROS

Dans les versions de Microsoft Office Excel 2003 et antérieur la majorité des outils de base sur les macros enregistrée se trouvent dans le menu **Outils/Macro**:



Depuis la version 2007 c'est un peu plus compliqué. Il faut idéalement activer un onglet dans le ruban qui se nomme l'onglet **Développeur**. Pour cela faites un clic droit n'importe où dans le ruban et choisissez l'option **Personnaliser le Ruben**:



Il vient alors:

Options Excel

Général
Formules
Vérification
Enregistrement
Langue
Options avancées
Personnaliser le Ruban
Barre d'outils Accès rapide
Compléments
Centre de gestion de la confidentialité

Personnalisez le Ruban.

Choisir les commandes dans les catégories suivantes :
Commandes courantes

Actualiser tout
Annuler
Aperçu avant impression et impri...
Augmenter la taille de police
Bordures
Calculer maintenant
Centrer
Collage spécial...
Coller
Coller
Connexions
Copier
Couleur de police
Couleur de remplissage
Couper
Courrier électronique
Définir
Enregistrer
Enregistrer sous
Figer les volets
Filtrer
Format de cellule...
Formes
Fusionner et centrer
Gestionnaire de noms
Image...
Impression rapide
Insérer des cellules...

Ajouter >>
<< Supprimer

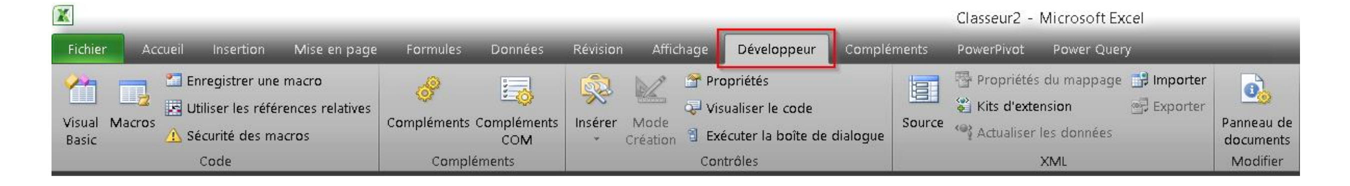
Personnaliser le Ruban :
Onglets principaux

Onglets principaux
Accueil
Presse-papiers
Police
Alignement
Nombre
Style
Cellules
Édition
Insertion
Mise en page
Formules
Données
Révision
Affichage
Développeur
Compléments
PowerPivot
Power Query
Search
Suppression de l'arrière-plan

Nouvel onglet
Nouveau groupe
Renommer...
Personnalisations : Réinitialiser
Importer/Exporter

OK
Annuler

Et donc il suffit de cocher l'option **Développeur** mise en évidence ci-dessus. Vous aurez alors un nouvel onglet à l'écran qui contient tout ce qu'il faut pour faire des macros enregistrées!

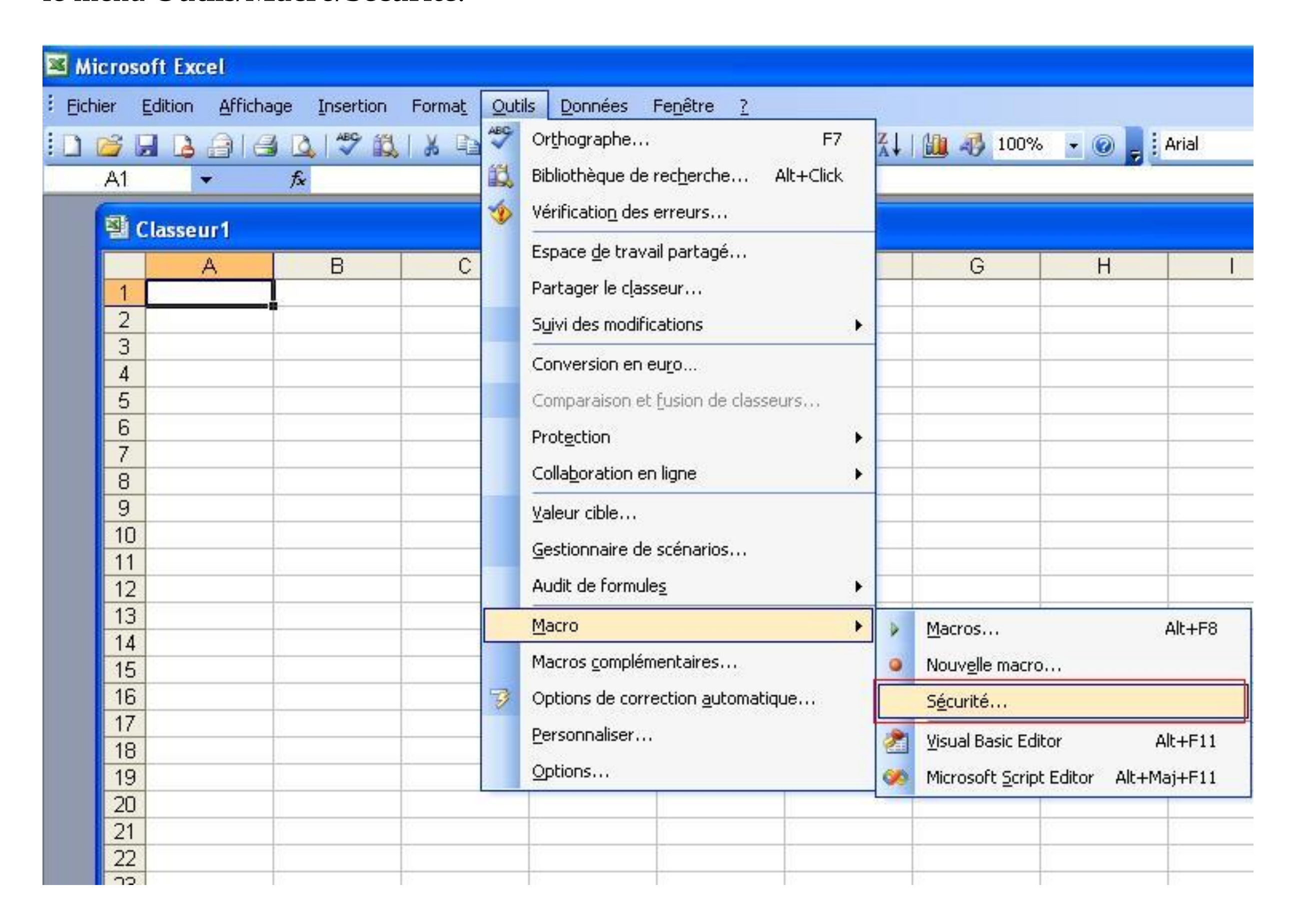


6. PROTECTION ET CERTIFICATION

6.1 Sécurité par défaut des macros

Par défaut dans la majorité des entreprises la sécurité des macros est telle qu'il est impossible de les exécuter et donc même de les tester (bien évidemment pour empêcher la propagation des macros-virus). Cette situation n'est pas acceptable dans une salle de formation et d'autant plus dans une formation sur les macros.

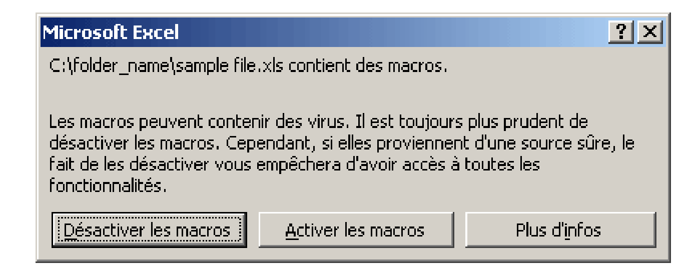
Pour changer le niveau de sécurité des macros dans les versions 2003 et antérieurs allez dans le menu **Outils/Macro/Sécurité**:



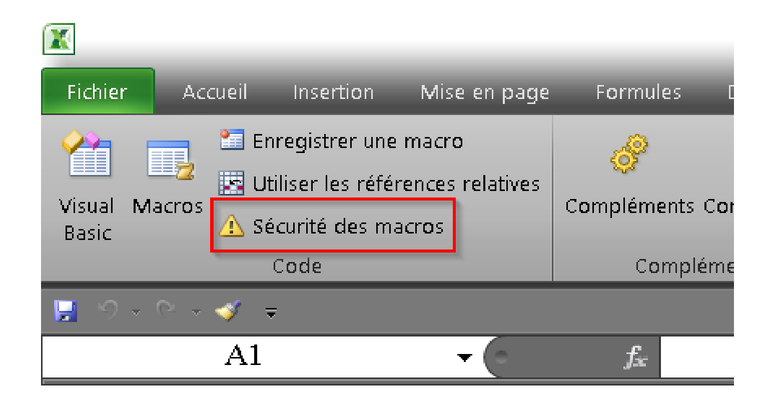
Vous aurez alors:



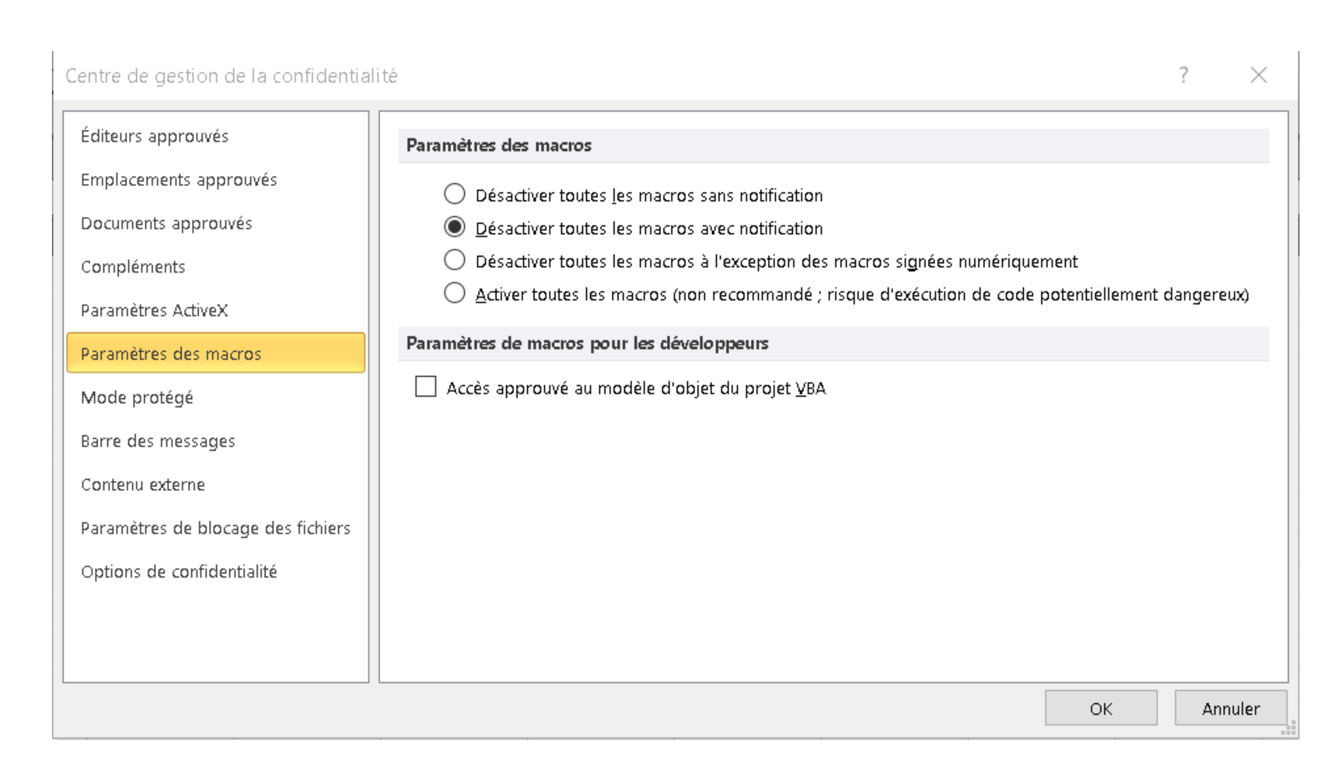
Idéalement dans la formation sélectionnez le niveau de sécurité dit **faible** (l'onglet sur les **Sources fiables** sera vu bien plus tard) et fermez/rouvrez le tableur pour que la modification soit prise en compte. Sinon quoi vous aurez à l'ouverture de chaque classeur contenant des macros le message suivant:



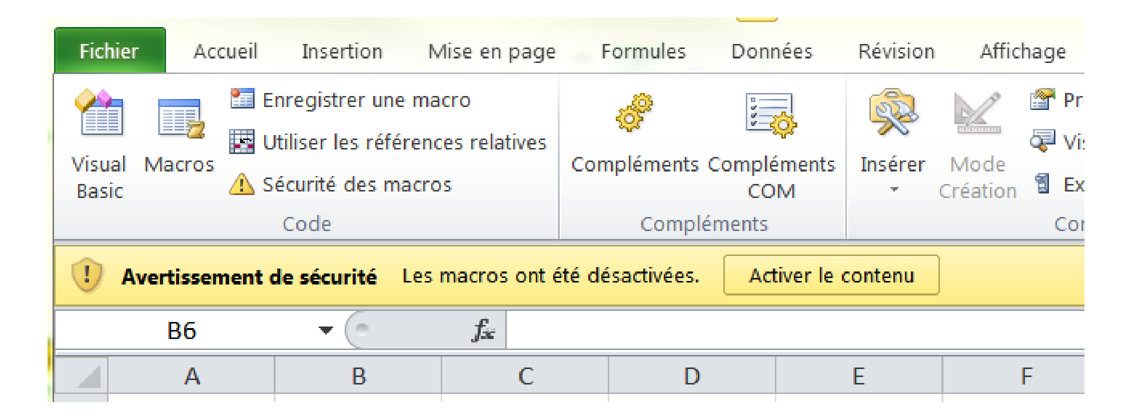
Depuis la version 2007 du tableur on peut passer par l'onglet **Développeur** pour faire simple en cliquant sur le bouton **Sécurité des macros**:



Il vient alors:



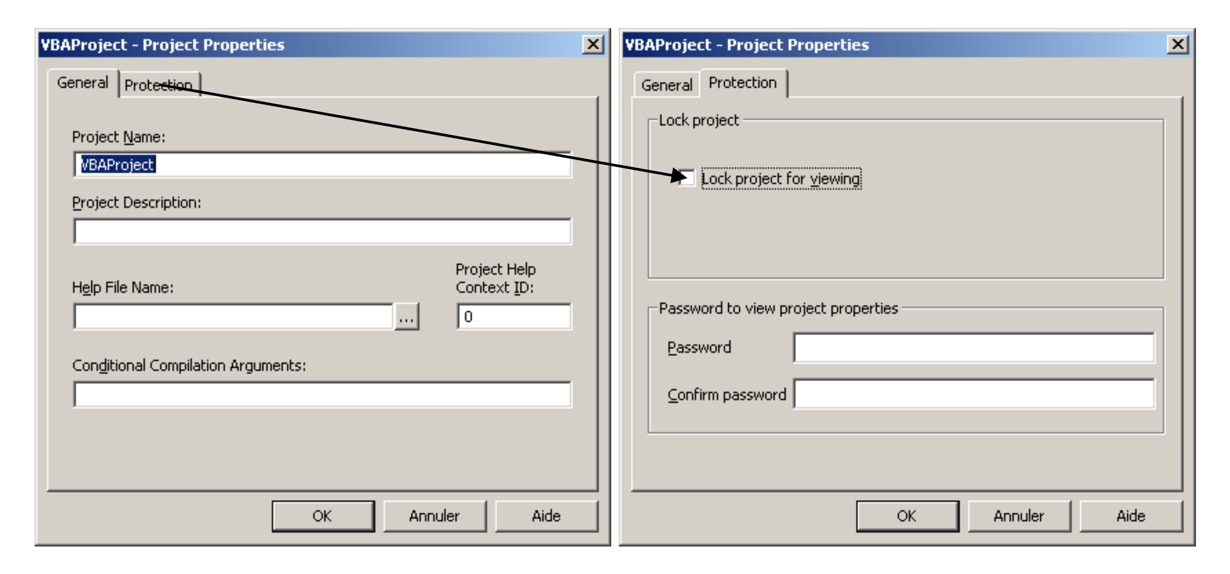
Dans le cadre de la formation il est conseillé de prendre la dernière options **Activer toutes les macros…**sinon en laissant le 2ème activé vous aurez à chaque ouverture d'un fichier contenant des macros le message suivant:



et fermez/rouvrez le tableur pour que la modification soit prise en compte.

6.2 Protection d'un projet

Il s'agit d'un exercice de style extrêmement simple. Pour protéger avec un mot de passe l'accès à votre code de projet, il suffit dans l'éditeur VBAE (**Alt+F11**) d'aller dans le menu **Outils / Propriétés du projet V.B.A.** et de saisir le mot de passe à l'endroit indiqué:



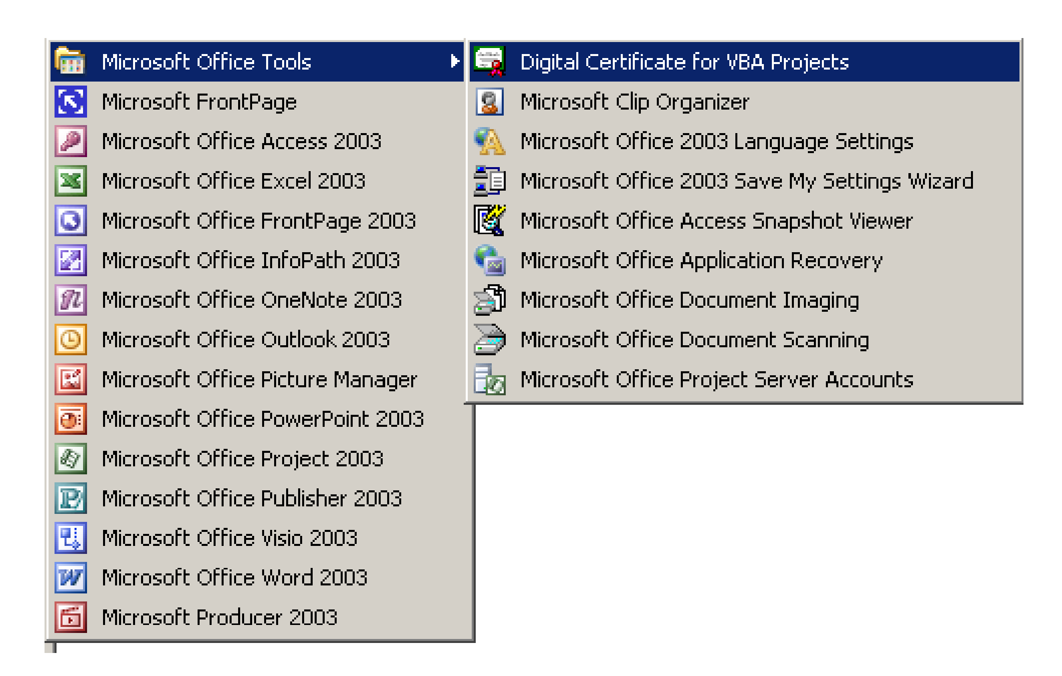
Il suffit ensuite de respectes les règles d'usage quant au choix du mot de passe pour qu'il ne soit pas crackable trop facilement.

Attention!!! Le mot de passe n'est pas pris en compte si le fichier est totalement vide!!!

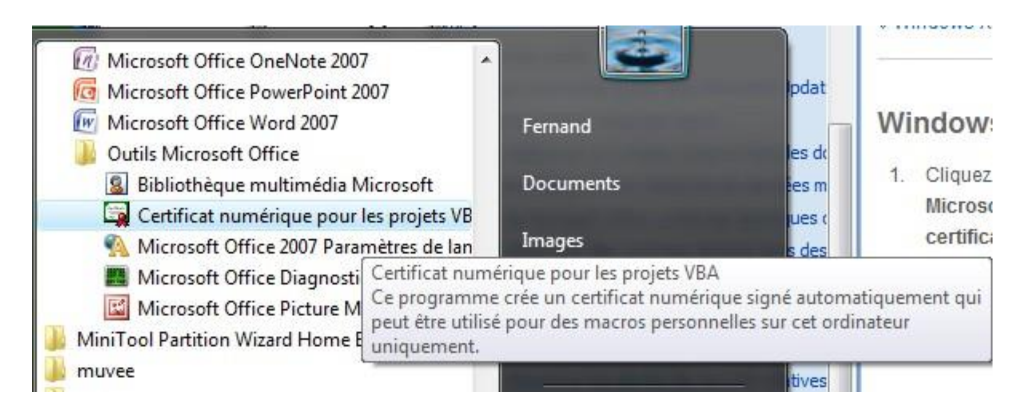
6.3 Signature d'un projet

Il s'agit ici d'authentifier (sans passer par un centre d'authentification racine) le projet afin que les ordinateurs ayant un niveau de sécurité moyen des macros puissent exécuter en toute confiance, sans validation les macros provenant d'une source donnée.

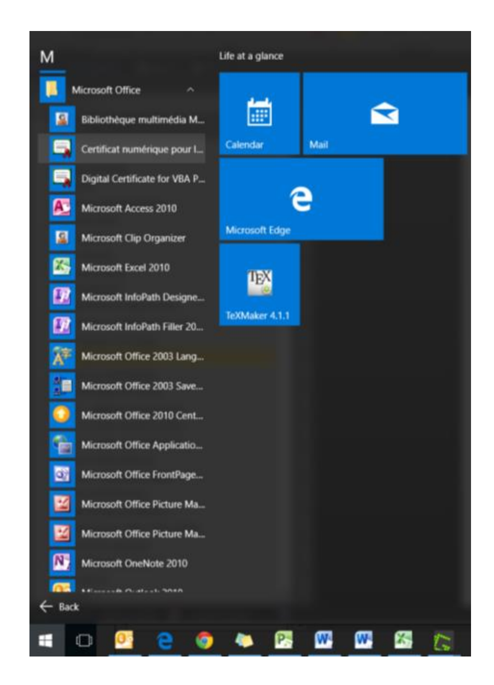
Avant tout chose, il faut créer un certificat. Pour ce faire, il suffit d'aller dans le menu **Démarrer** exécuter l'application complémentaire ***Digital Certificate for V.B.A. Projects***:



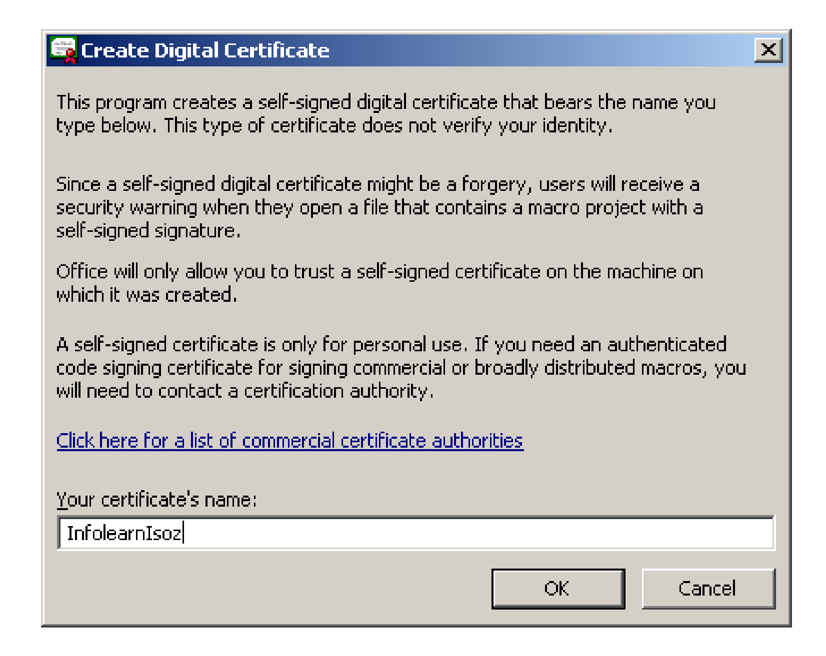
…ou pour ceux qui ont la version français avec le menu **Windows** (Windows Vista, Windows Seven):



…ou pour ceux qui ont Windows 10:



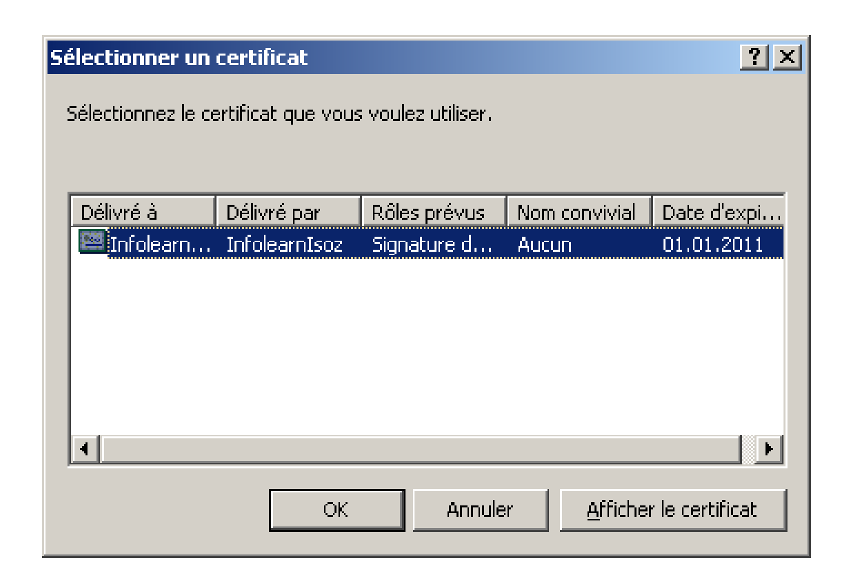
Par la suite, il faut saisir le nom du certificat:



Après avoir cliqué sur **OK**, dans le VBAE vous devez aller dans le menu **Outils/Signature digitale**:

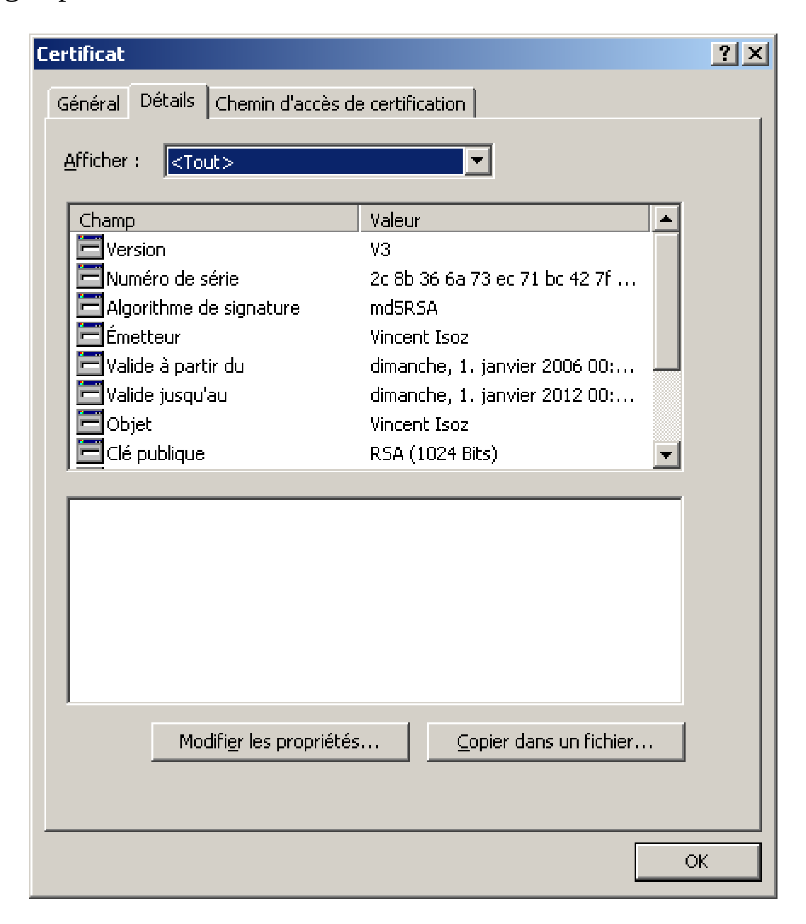


Il suffit de cliquer sur **Choose**:



Vous pouvez toujours cliquer sur **Afficher le certificat** mais les informations sont un peu trop technique pour être abordées dans le cadre de ce cours. Vous pouvez cependant cliquer sur **OK**.

Il faut ensuite exporter ce certificat pour l'envoyer par e-mail à votre collègue, joint avec votre fichier MS Excel contenant la ou les macros. Pour cela, cliquez sur **Afficher le certificat** dans la boîte de dialogue précédente:



Cliquez sur le bouton **Copier dans un fichier…** et ensuite cliquez quatre fois sur **Suivant**:

Assistant Exportation de certificat

Bienvenue !

Cet Assistant vous aide à copier des certificats, des listes de certificats de confiance et des listes de révocation de certificats depuis le magasin de certificats vers votre disque dur.

Un certificat, émis par une Autorité de certification, est une confirmation de votre identité et contient des informations utilisées pour protéger vos données ou établir des connexions réseau sécurisées. Le magasin de certificats est la zone système où les certificats sont conservés.

Pour continuer, cliquez sur Suivant.

< Précédent | Suivant > | Annuler

Assistant Exportation de certificat

Exportation de la clé privée

Vous pouvez choisir d'exporter la clé privée avec le certificat.

Les clés privées sont protégées par mot de passe. Pour pouvoir exporter la clé privée avec le certificat, vous devez entrer son mot de passe dans une des pages suivantes.

Voulez-vous exporter la clé privée avec le certificat ?

- Oui, exporter la clé privée
- Non, ne pas exporter la clé privée

Remarque : la clé privée associée est marquée comme non exportable. Seul le certificat peut être exporté.

< Précédent | Suivant > | Annuler

Assistant Exportation de certificat

Format de fichier d'exportation

Les certificats peuvent être exportés sous plusieurs formats de fichier.

Sélectionnez le format à utiliser :

- Binaire codé DER X.509 (.cer)
- Codé à base 64 X.509 (.cer)
- Standard de syntaxe de message cryptographique - Certificats PKCS #7 (.p7b)
  - Inclure tous les certificats dans le chemin d'accès de certification si possible
- Échange d'informations personnelles - PKCS #12 (.pfx)
  - Inclure tous les certificats dans le chemin d'accès de certification si possible
  - Activer la protection renforcée (nécessite IE 5.0, NT 4.0 SP4 ou supérieur)
  - Supprimer la clé privée si l'exportation s'est terminée correctement

< Précédent | Suivant > | Annuler

Assistant Exportation de certificat

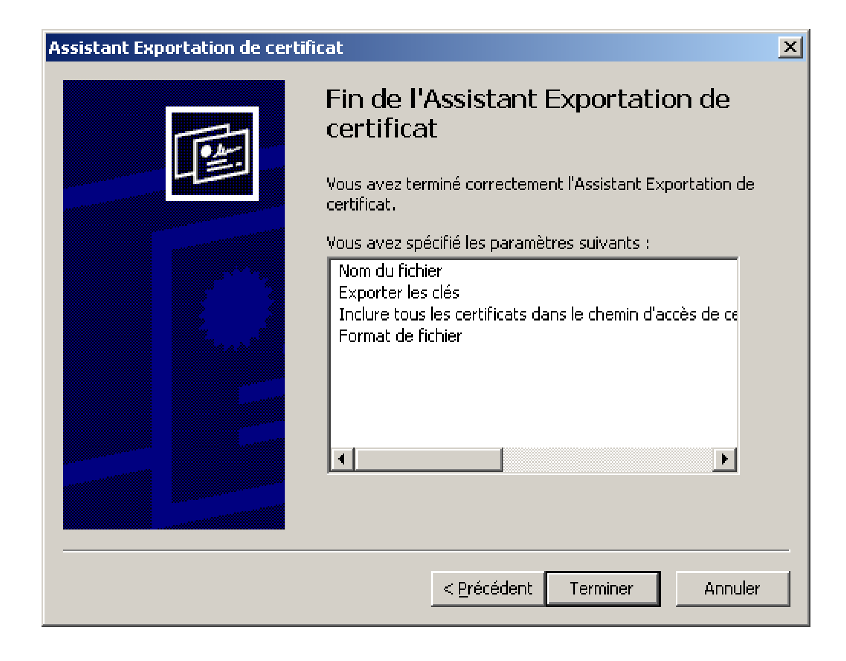
Fichier à exporter

Spécifiez le nom du fichier à exporter

Nom du fichier : | Parcourir...

< Précédent | Suivant > | Annuler

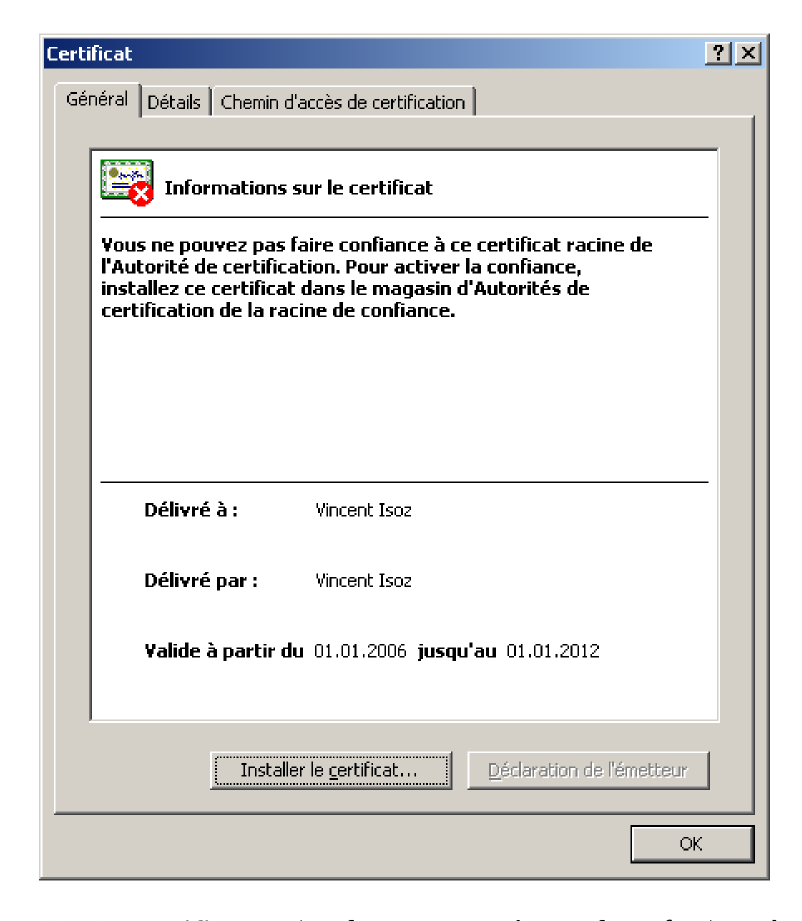
Dans **Nom du fichier**, saisissez votre nom et prénom et cliquez ensuite sur **Suivant**:



et ensuite sur **Terminer** (le certificat sera enregistré par défaut dans le dossier *Mes documents*).

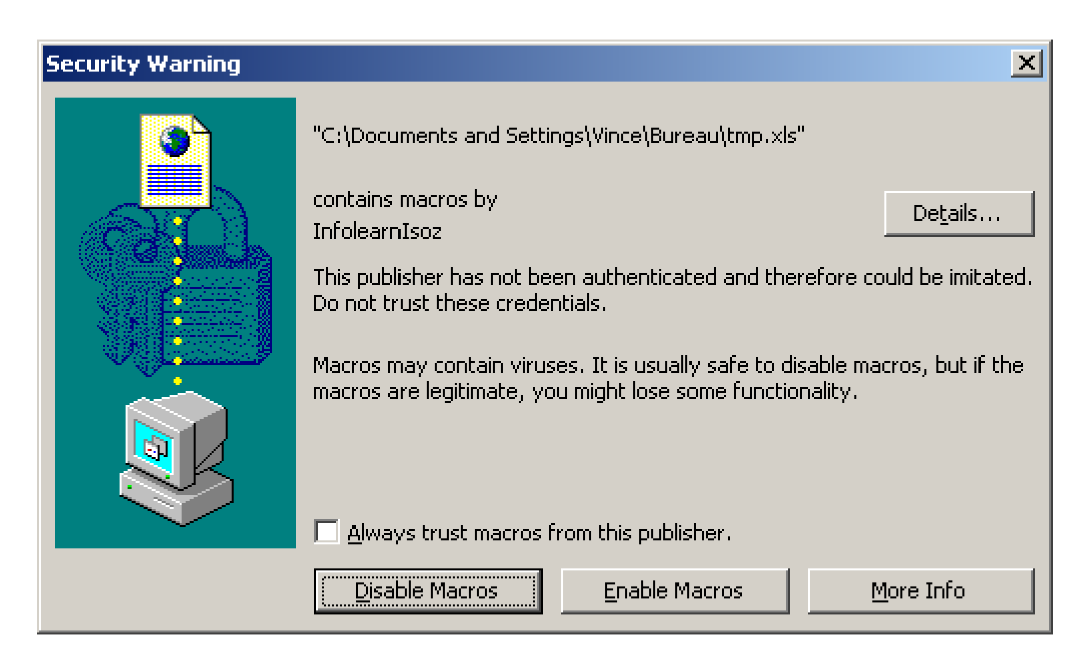


Lors de l'envoi de la réception de ce certificat, tout utilisateur devra faire un double clic dessus:

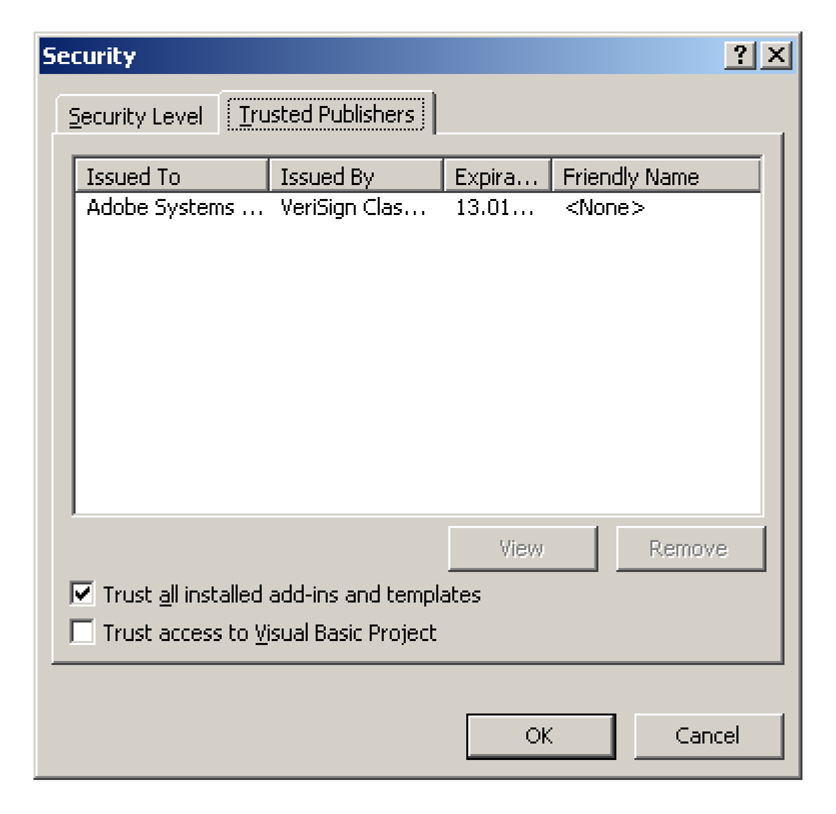


et cliquer sur **Installer le certificat…** (et cliquer sur **Suivant** deux fois) après quoi seulement il pourra ouvrir le classeur avec le projet V.B.A. signé par cette même signature et la reconnaître comme source sûre.

Ensuite, tout utilisateur ayant le niveau de sécurité de macros activé sur **Moyen** et ouvrant le projet aura à l'écran le message suivant:



L'objectif étant bien évidemment de cliquer sur la case à cocher si la source est "sûre". L'utilisateur peut ensuite vérifier les sources sûres installées sur sa machine en allant dans **Outils/Macros/Sécurité** et si besoin les supprimer de la liste Excel:



Pour supprimer un certificat que vous avez créé dans votre système Windows il faut passez par les options d'Internet Explorer **Outils/Options Internet**, onglet **Contenu** et cliquer sur le bouton **Certificats**.

7. MACROS ENREGISTRÉES

MS Excel, comme MS Word ou MS PowerPoint, etc. possèdent un outil génial: l'enregistreur de macros. Il crée une macro et transforme en langage V.B.A. (bon le code est dégueulasse mais cela fonctionne quand même la plupart du temps) toutes les commandes effectuées par l'utilisateur dans l'application hôte. Il permet d'automatiser sans aucune connaissance de la programmation certaines de vos tâches et également de se familiariser avec le langage V.B.A.

Dans le monde des macros nous différencions dans un premier temps (dans l'ordre d'apprentissage habituel):

1. Les **macros absolues** qui vont répéter des actions toujours et exactement sur les mêmes cellules.

2. Les **macros relatives** qui sont capable d'effectuer des actions à des positions variables relativement à un point de référence.

3. Les **macros personnelles** qui sont des macros qui peuvent très facilement être exécutées sur tous vos classeurs *.xls (anciens ou nouveaux).

4. Les **macros événementielles** qui sont des macros s'exécutant automatiquement à l'ouverture ou à la fermeture d'un fichier *.xls.

5. Les **macros complémentaires** *.xla qui sont des macros partagées en plusieurs collaborateurs ou plusieurs départements.

6. Les **macros référencées** qui sont des macros appelées d'un code V.B.A. d'un autre fichier.

Voyons un bref exemple par écrit de chacun (évidemment dans la formation nous en verrons beaucoup plus!).

Attention!! Comme le confirme Microsoft sur son propre blog:

https://blogs.office.com/2011/11/22/when-a-macro-wont-cut-it-try-a-vba-script/

Une macro n'est pas un script VBA! Donc il ne faut pas dire que c'est la même chose!!!!!!

Et gardez en tête que les macros ont les problèmes connus suivants:

- Le code généré par la macro enregistrée est souvent mal structuré et donc non optimal

- Le code géneré par la macro enregistrée est souvent plein de lignes à double et donc non optimal

- Le code généré par la macro enregistrée contient souvent des lignes inutiles et est donc non optimal

- Le code généré par la macro enregistrée ne peut pas se générer dès que vous enregitrez quelque chose en-dehors de Microsoft Excel

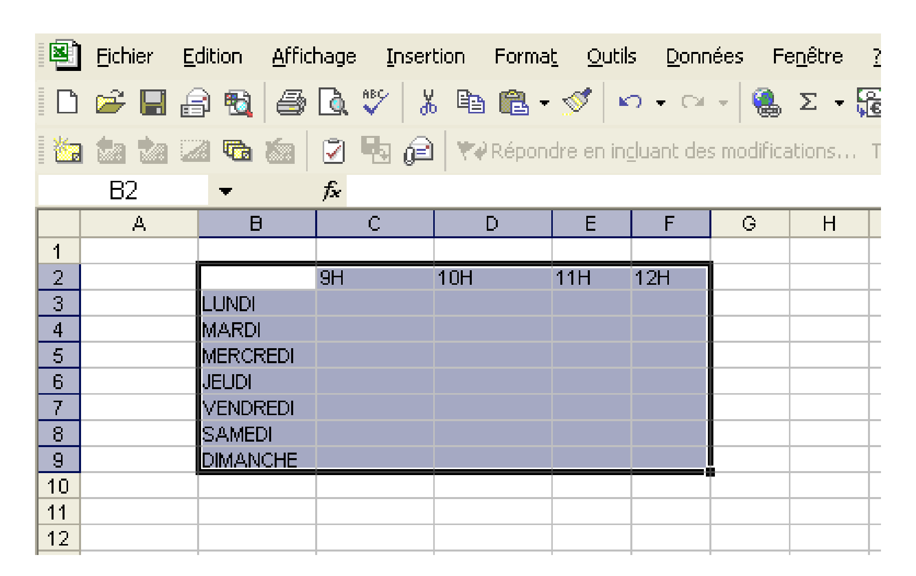
- Le code généré par la macro enregistrée ne peut pas générer des boucles
- Le code généré par la macro enregistrée ne peut pas générer des interaction avec l'utilisateur
- Le code généré par la macro enregistrée ne peut pas s'adapter à un contexte qui n'est parfaitement identique à celui qui était présent lors de l'enregistrement
- Le code généré par la macro enregistrée ne peut créer des varables
- Le code généré par la macro enregistrée ne gérer les erreurs de façon pro-active

Ceci étant donc dit et connu, pratiquons-les un peu et voyons leurs limites.

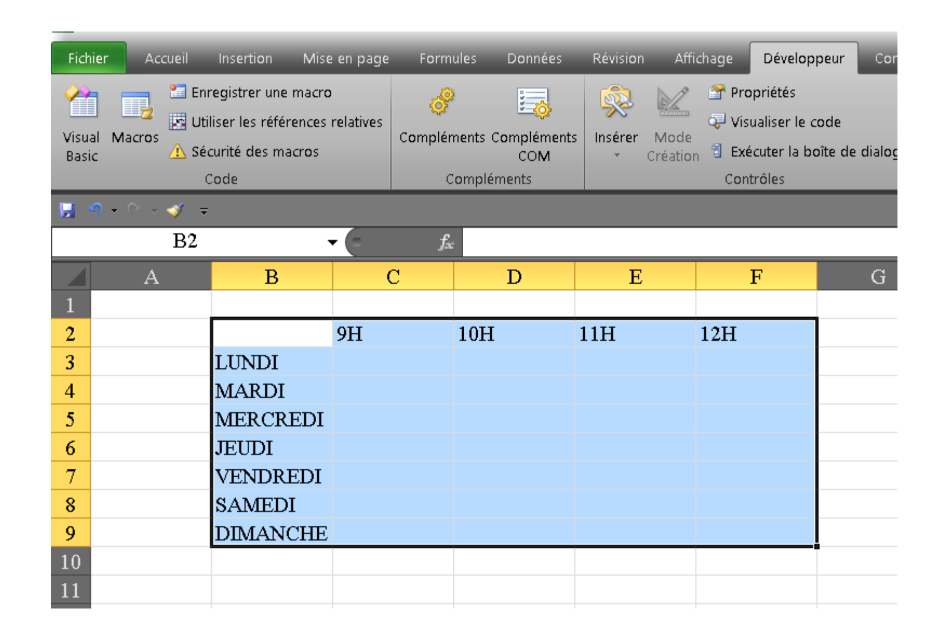
7.1 Macros absolues

Nous allons mettre au même format tous les tableaux que nous créons. Plutôt que de reproduire à chaque fois les mêmes commandes, nous allons utiliser l'enregistreur de macros.

Nous sélectionnons d'abord notre tableau. En effet, si vous le sélectionnez après avoir lancé l'enregistrement, la macro reproduirait cette sélection à chaque lancement et s'exécuterait toujours sur les mêmes cellules.



Ce qui ressemblera dans les versiona 2007 et ultérieures à:

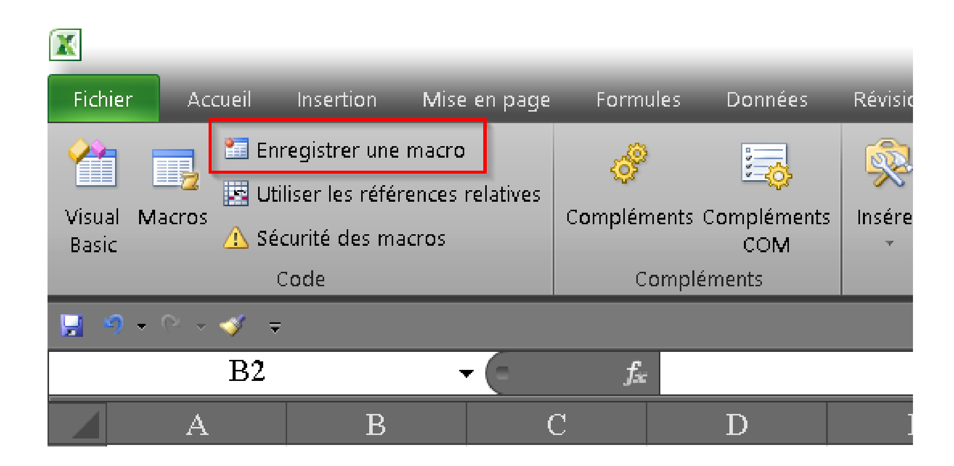


Nous lançons l'enregistrement par le menu **Outils/Macros/Nouvelle macro** pour MS Excel 2003 et antérieur et il vient:



Remarque: Dans la pratique il est recommandé de commencer toutes vos macros par l'abréviation mcr_ et des les nommer en anglais. Et aussi de se renseigner si un charte de création des macros existe dans votre entreprise!

Pour MS Excel 2007 et antérieur nous cliquerons sur:



Nom de la macro: Il est important de renommer de façon explicite la macro. Le nom de la macro doit commencer par une lettre et ne doit pas contenir d'espaces. Utilisez le caractère de soulignement pour séparer les mots (normalement il faut respecter des normes que nous verrons plus tard)

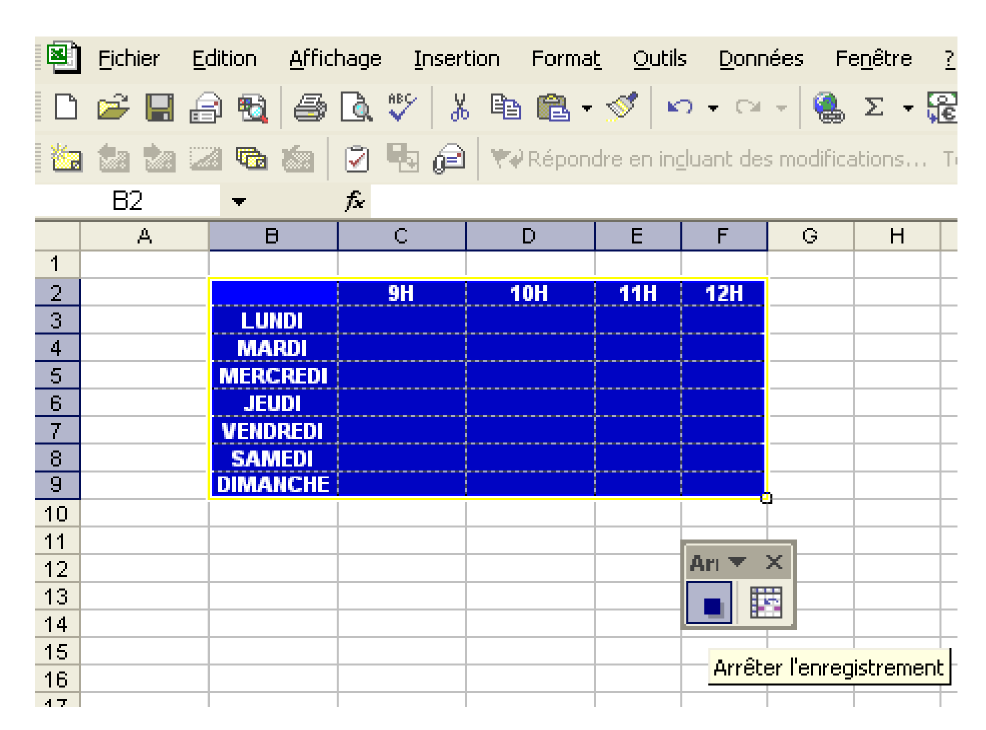
Remarque: Si vous nommez votre macro Auto_Open ou Auto_Close celle-ci s'exécutera automatiquement à l'ouverture ou respectivement la fermeture de votre classeur !!

Touche de raccourci: Il est possible de créer un raccourci clavier pour lancer la macro en saisissant une lettre. Vous pouvez utiliser les majuscules et lancer la macro par les touches **Ctrl+MAJ+Lettre**.

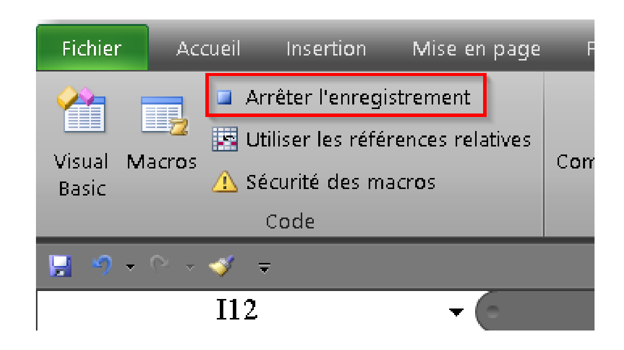
Enregistrer la macro dans: Classeur contenant la macro. La macro ne pourra ensuite s'exécuter que si le classeur dans lequel la macro a été enregistrée est ouvert !!! Si vous choisissez "classeur de macros personnelles", un nouveau classeur est créé et enregistré dans le dossier "xlstart" de façon que vos macros soient disponibles à chaque fois que vous utilisez MS Excel.

Description: Vous pouvez donner une description à votre macro.

Après avoir cliqué sur **OK**, la macro va essayer au mieux d'enregistrer toutes les commandes que nous allons utiliser. Ici, nous changeons le format de notre tableau (Couleur, Polices, Encadrement).

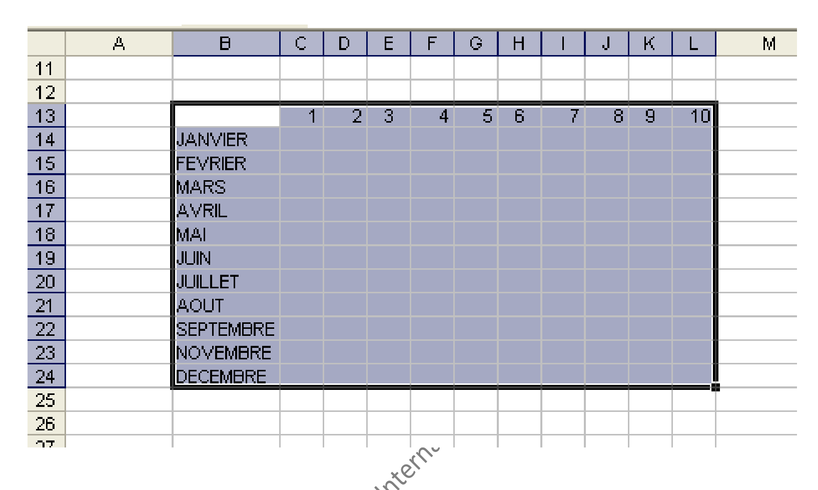


Nous arrêtons ensuite l'enregistrement en cliquant sur le bouton **Arrêter l'enregistrement** ou par le menu **Outil/Macro/Arrêter l'enregistrement** ou depuis MS Excel 2007:

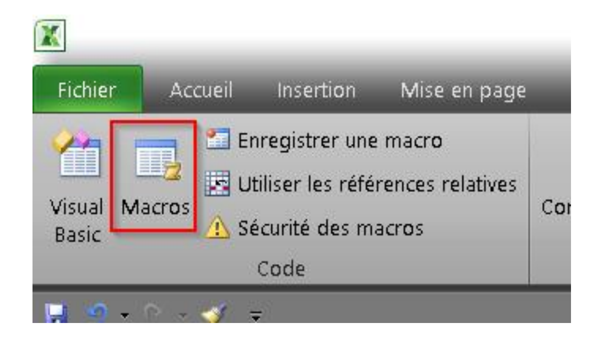


La macro est maintenant créée!!

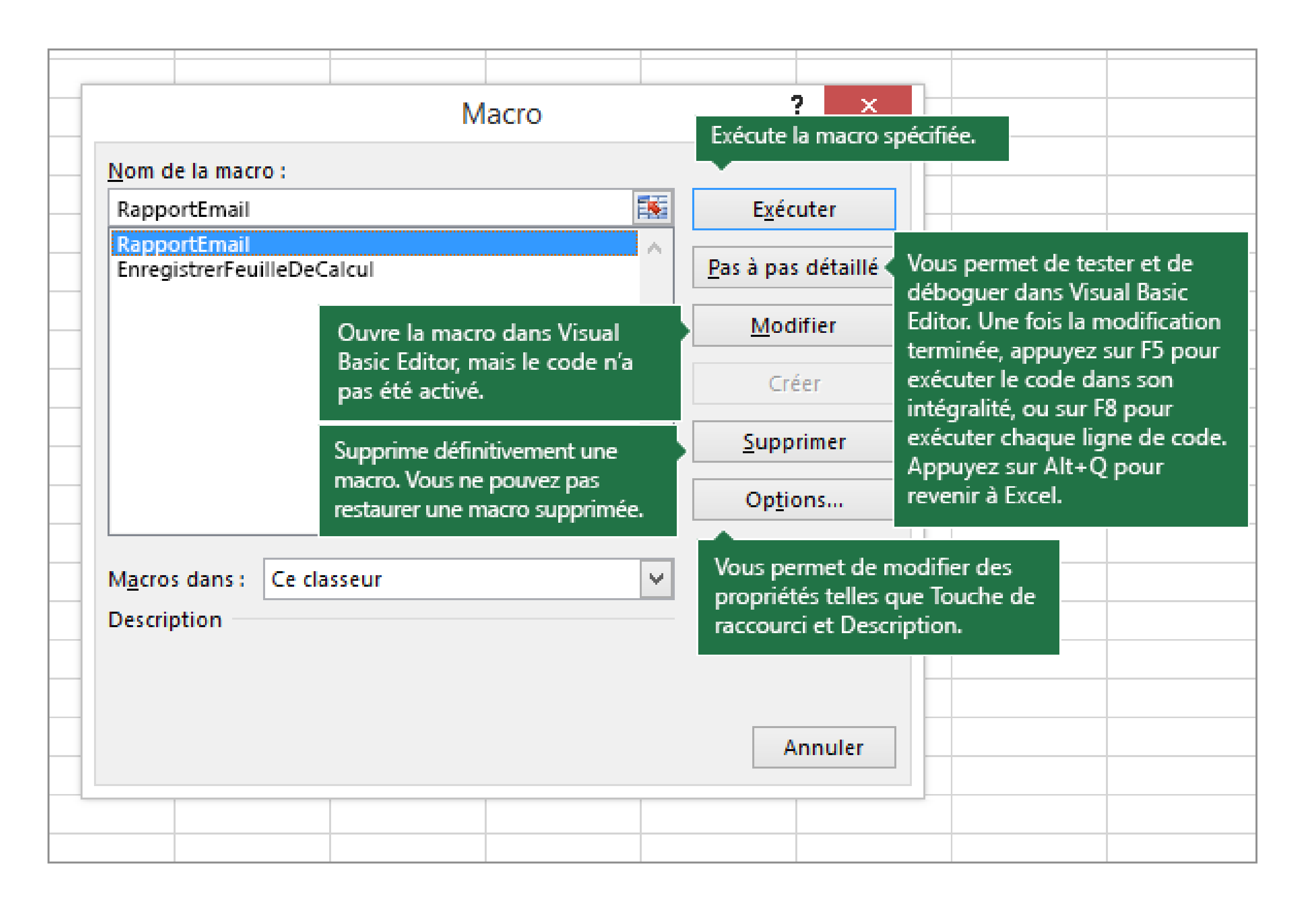
Nous avons ensuite un nouveau tableau sur lequel nous voulons reproduire le même format:



Nous sélectionnons notre nouveau tableau et alors utiliser la macro précédemment créée (à condition que le classeur dans laquelle elle a été enregistrée soit ouvert). Nous la lançons par le menu **Outils/Macro/Macros** (ou **Alt+F8**) ou depuis MS Excel 2007 via:

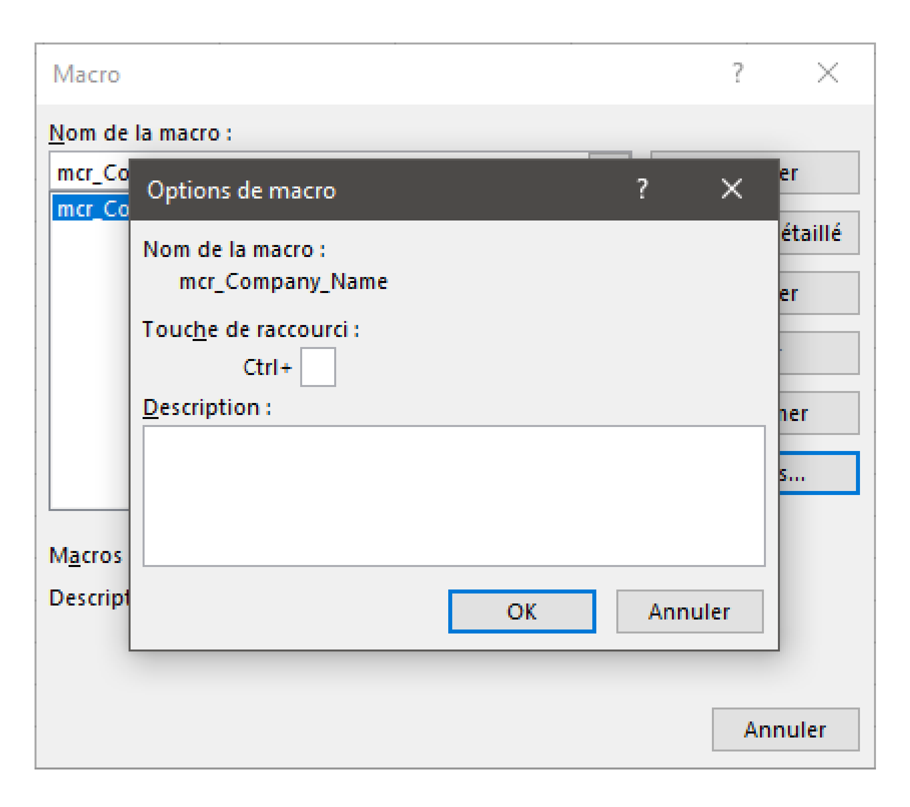


Et il vient:



Notez que c'est depuis ici que vous pouvez simplement supprimer une Macro en cliquant sur le bout **Supprimer** (…).

Et que nous pouvons changer le raccourci ou la description de la Macro en cliquant sur **Options…**:



Nous sélectionnons la macro désirée puis cliquons sur **Exécuter**:

	1	2	3	4	5	6	7	8	9	10
JANVIER										
FEVRIER										
MARS										
AVRIL										
MAI										
JUIN										
JUILLET										
AOUT										
SEPTEMBRE										
NOVEMBRE										
DECEMBRE										

Nous aurions également pu lancer la macro par le raccourci clavier **Ctrl+F** que nous avions défini lors de sa création.

L'enregistreur de macros est donc un outil très pratique.

Quelques précautions à prendre lors de son utilisation cependant :

- Bien penser aux commandes que nous voulons enregistrer pour éviter de créer du code inutile.

- Savoir dans quel classeur enregistrer la macro pour son utilisation future.

- Renommer les macros de façon explicite.

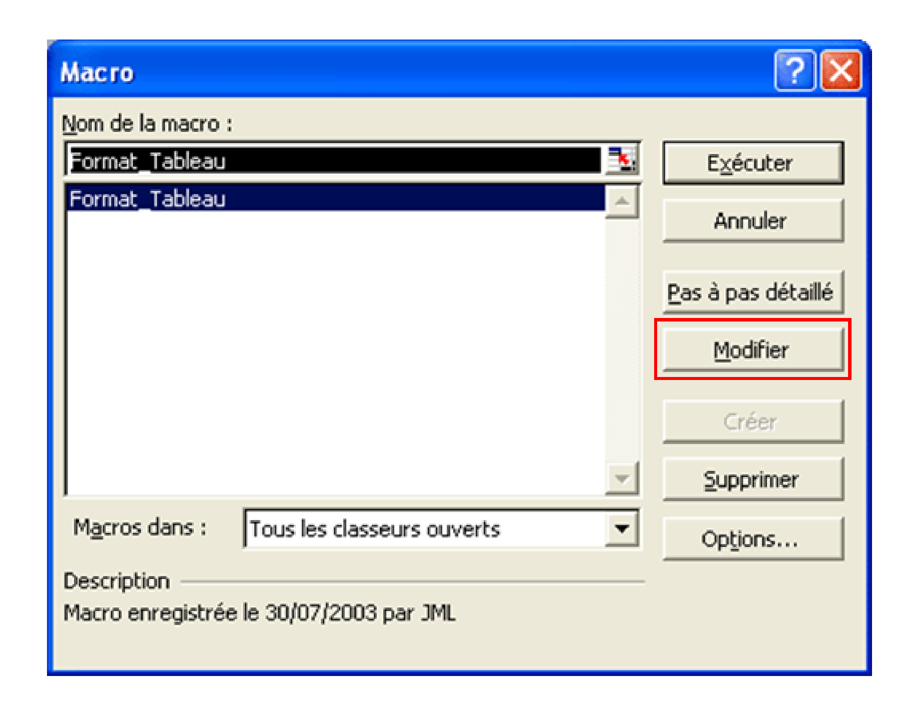
Limites de l'enregistreur de macros:

- Il ne peut que traduire les commandes réalisées par l'utilisateur dans l'application hôte et écrit en général plus de lignes de code que nécessaire.

- Lors de l'écriture de macros, nous verrons comment utiliser et réduire le code crée par l'enregistreur.

- Il n'enregistre de loin pas tout et pas correctement

Les macros créées sont visibles et modifiables via l'éditeur de macro VBAE (Visual Basic Application Editor) comme nous le verrons lors de notre étude du V.B.A ou en cliquant sur le bouton Modifier visible ci-dessous:



Ce qui donne:

```
'********************************
' Ceci est un exemple de macro automatique utilisé pour la recherche de
' commande de tri, d'affichage de l'outil grille et de tri à nouveau!
' Vous observerez qu'il s'agit d'une procédure et non d'une fonction!
' De plus, le code est "sale" et non optimal
'********************************

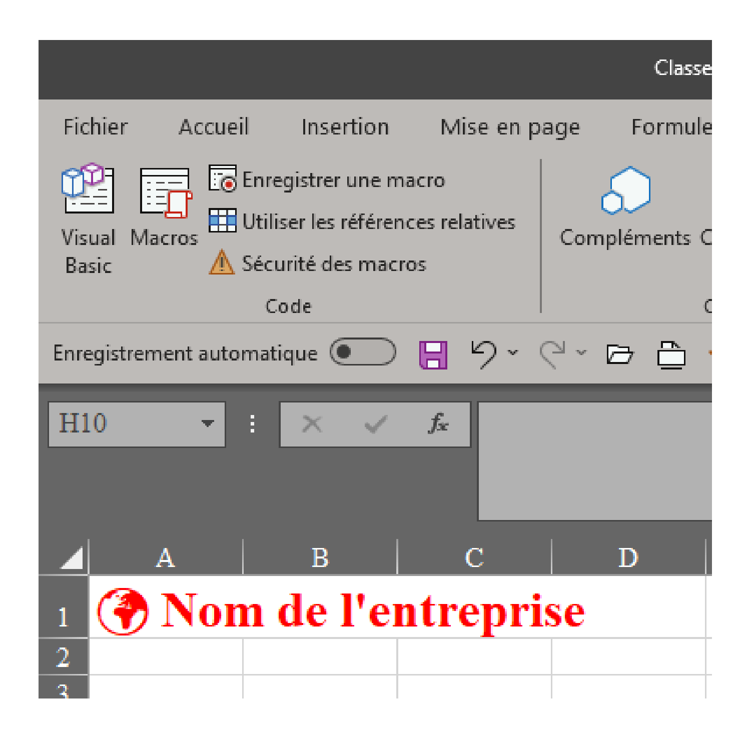
Sub Macro1()

' Macro1 Macro
' Macro enregistrée le 20.02.2002 par Vincent ISOZ

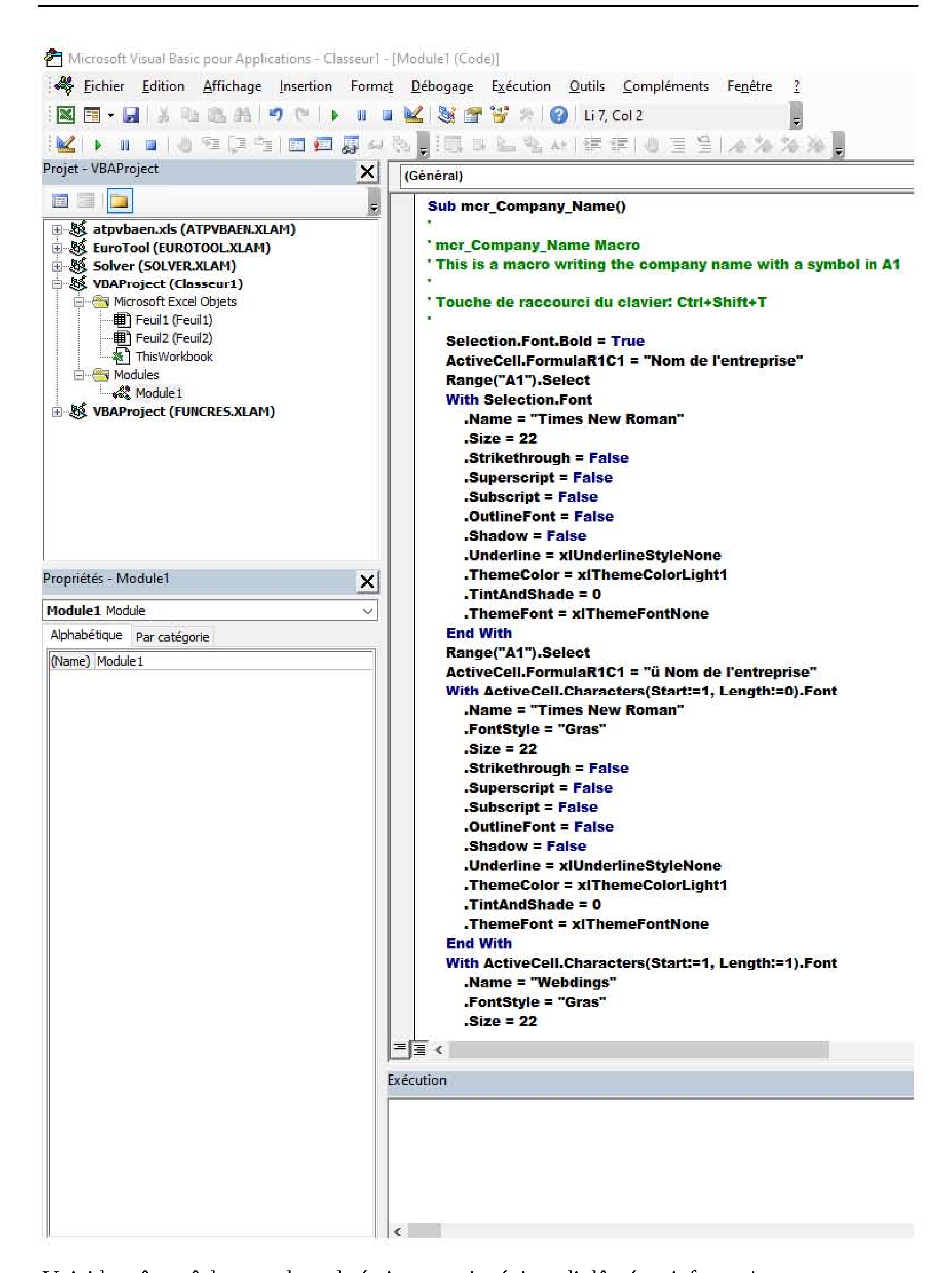
    Range("A1:J110").Select
    Selection.Sort Key1:=Range("A16"), Order1:=xlAscending,
Header:=xlGuess, _
        OrderCustom:=1, MatchCase:=False, Orientation:=xlTopToBottom
    ActiveSheet.ShowDataForm
    Selection.Sort Key1:=Range("A16"), Order1:=xlAscending,
Header:=xlGuess, _
        OrderCustom:=1, MatchCase:=False, Orientation:=xlTopToBottom
    ActiveWindow.SelectedSheets.PrintPreview
End Sub
```


7.1.1 Nettoyage de Macro

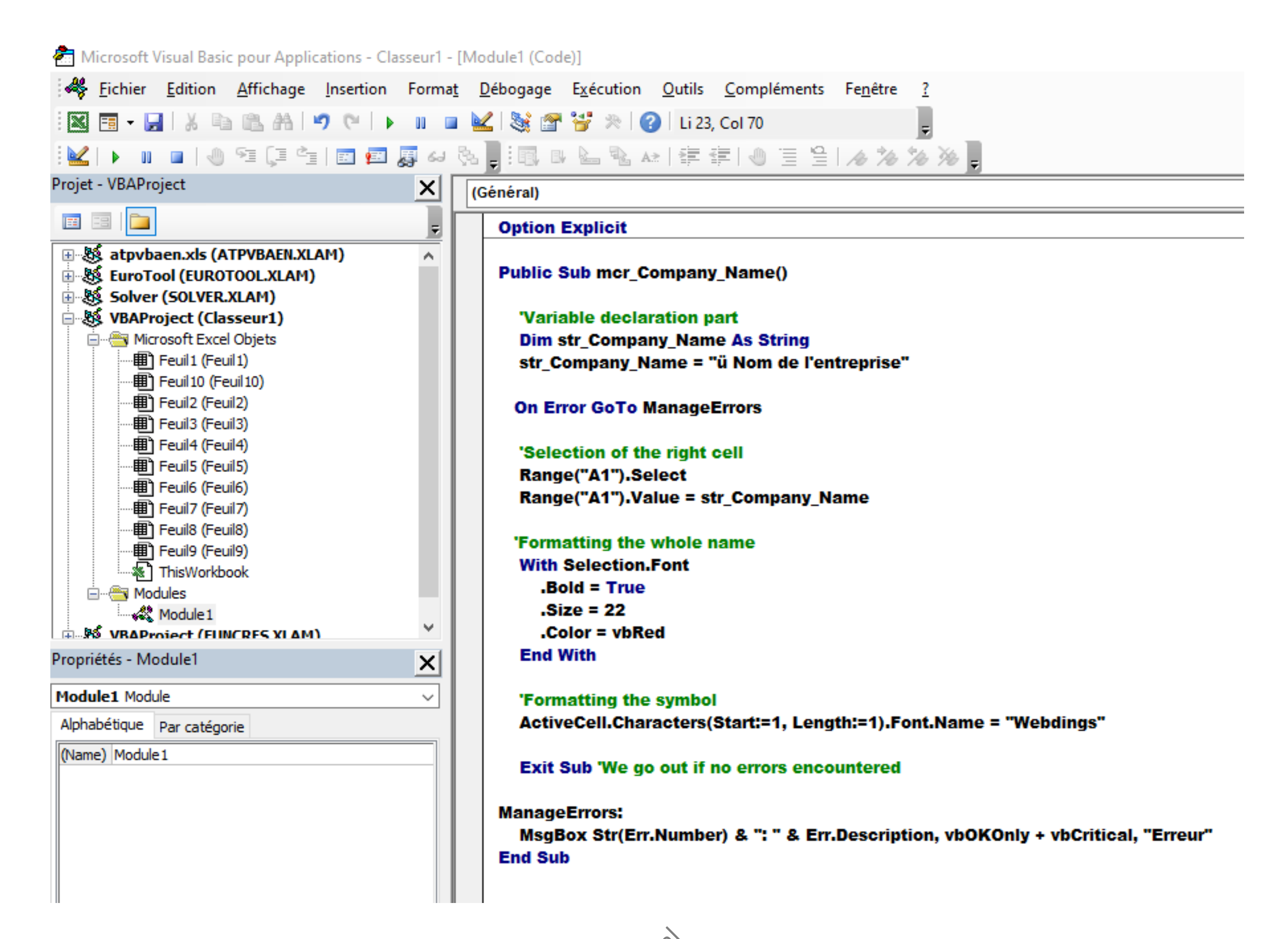
Parlons maintenant du problème de la propreté du code d'un macro. Considérons un Macro extrêmement simple comme celle consitant à écrire un texte dans la cellule A1 de la feuille active.



Voici le code résultant qui ne tient même pas sur une capture d'écran complète…:



Voici la même tâche avec le code écrit par un ingénieur diplômé en informatique:



Évidemment il faut souvent des années de pratique pour:

1. Savoir ce qu'il faut nettoyer
2. Savoir écrire des commentaires qui font du sens
3. Savoir gérer des erreurs de manière pro

Bref… écrirer et savoir faire du code prend souvent autant de temps que d'apprendre un langue étrangère.

7.2 Macros relatives

Nous l'avons vu précédemment nous avons dû sélectionner d'abord le tableau avec de lance l'enregistrement de la macro sinon quoi elle s'appliquait toujours sur la même zone. Ce comportement dit "absolu" pose de problème dans de nombreuses situations et nous allons en voir un petit exemple qui peut bien évidemment être compliqué à l'infini.

Considérons le tableau suivant:

	A	B
1	Année	C.A.
2	2003	32200
3	2004	33088
4	2005	40000
5	2006	53080

En suivant la même procédure qu'avant faites en sorte avec une macro que la somme arithmétique des C.A. apparaisse dans la somme arithmétique:

	A	B
1	Année	C.A.
2	2003	32200
3	2004	33088
4	2005	40000
5	2006	53080
6		158368

Ensuite ajoutez une nouvelle année:

	A	B
1	Année	C.A.
2	2003	32200
3	2004	33088
4	2005	40000
5	2006	53080
6	2007	69070

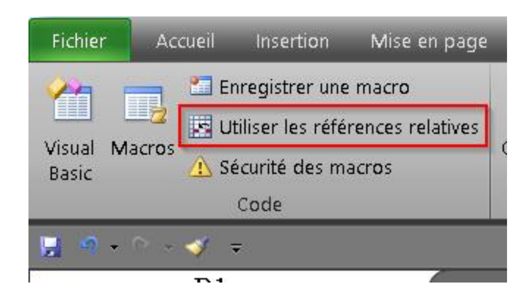
Et réexécutez la macro avec la méthode vue précédemment:

	A	B
1	Année	C.A.
2	2003	32200
3	2004	33088
4	2005	40000
5	2006	53080
6	2007	158368

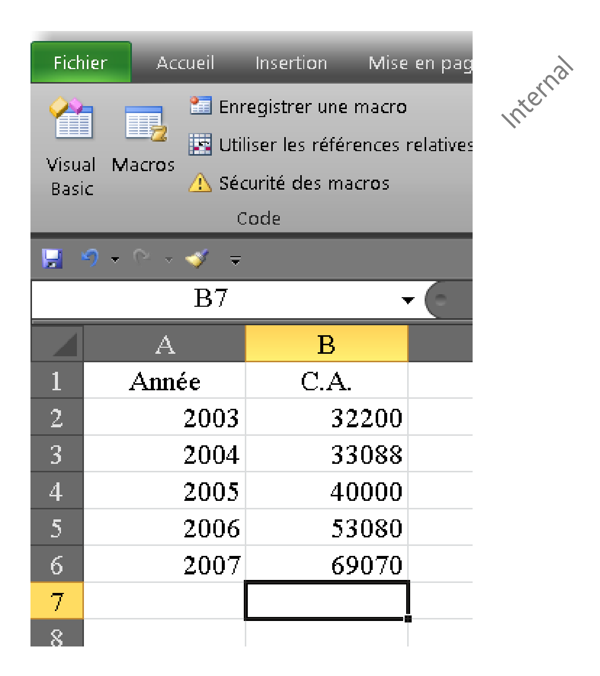
Vous pouvez effectivement constater que la somme se récrité toujours dans la même cellule… Cela provient du fait que la macro est par défaut en mode **absolu**!

Mantenant si vous reproduisez la procédure en mode relatif voici comment procédéer:

1. Lancer l'enregistrement de la macro comme à l'habitude
2. Positionnez-vous sur **B1**
3. Et cliquez maintenant sur **Utiliser les références relatives**:



4. Déplacez-vous en bas de la colonne B avec **Ctrl+Flèche Bas** + une fois le bouton flèche vers le bas pour vous positionner ainsi:



5. Et utilisez le bouton somme automatique:

	A	B	C
1	Année	C.A.	
2	2003	32200	
3	2004	33088	
4	2005	40000	
5	2006	53080	
6	2007	69070	
7		=SOMME($B$2:B6)	
8			

et n'oubliez pas de mettre les $ sur **B2**!!!

6. Validez par **Entrée** et arrêtez l'enregistrement de la macro.

7. Testez votre maco et observez que oui… elle s'écrire toujours au bon avec la bonne formule (sauf erreur de votre part ou problème d'installation du logiciel).

Attention!!! Le bouton **Utiliser les références relatives** reste actif entre chaque session de MS Excel donc prenez garde à son état avant d'enregistrer une nouvelle macro.

7.3 Interfaçage des macros

Pour exécuter les macros vous avez plusieurs possibilités que vous pouvez proposer à vos utilisateurs:

1. En faisant Alt+F8 mais vos utilisateurs apprécieront peu cette méthode. D'ailleurs cette méthode ne marche plus quand vous faites des macros complémentaires (voir plus loin).

2. En faisant des boutons sur les feuilles directement. C'est LA méthode à utiliser lorsque vous avez enregistré des macros en mode dans « Ce classeur ». Il existe deux méthodes pour affecter des macros à des boutons:

    a. En utilisant la barre d'outils **Formulaires**. C'est la méthode la plus simple au niveau du cours Macro mais pas la plus pro.

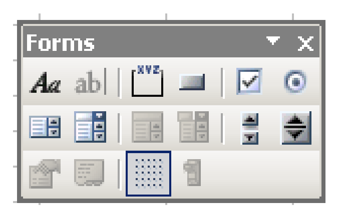
    b. En utilisant la barre d'outils **Boîte à outils contrôle**. C'est une méthode pro mais trop compliquée tant que nous n'aurons pas abordé la partie V.B.A. de ce support.

    c. En créant des barres d'outils. C'est une méthode que vous pouvez utiliser pour les macros enregistrées dans *Ce classeur* (mais normalement ce n'est pas le but) et c'est une méthode que vous êtes obligé d'utiliser pour les macro personnelles et les macros complémentaires.

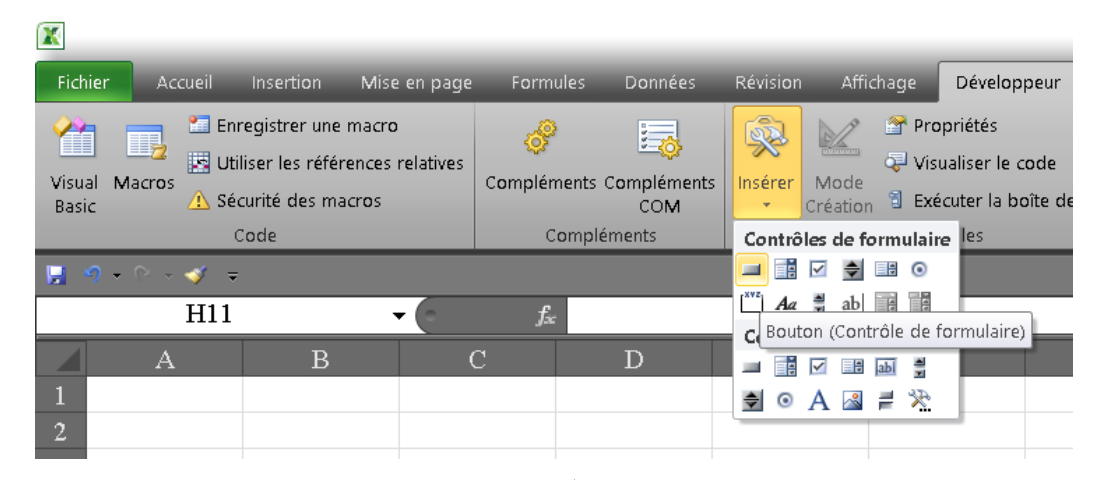
7.3.1 Éléments de formulaires

Comme nous l'avons précisé tout à l'heure, les éléments de formulaires sont utilisés lorsque vous commencez à apprendre les macros automatiques et que vous souhaitez créer un accès simple à l'exécution de macros qui ont été enregistrées dans « Ce classeur ».

Comment cela fonctionne-t-il? D'abord dans MS Excel 2003 et antérieure il faut activer une barre d'outils (cela est connu puisque vu au cours MS Excel avancé) qui se nomme **Formulaires**:

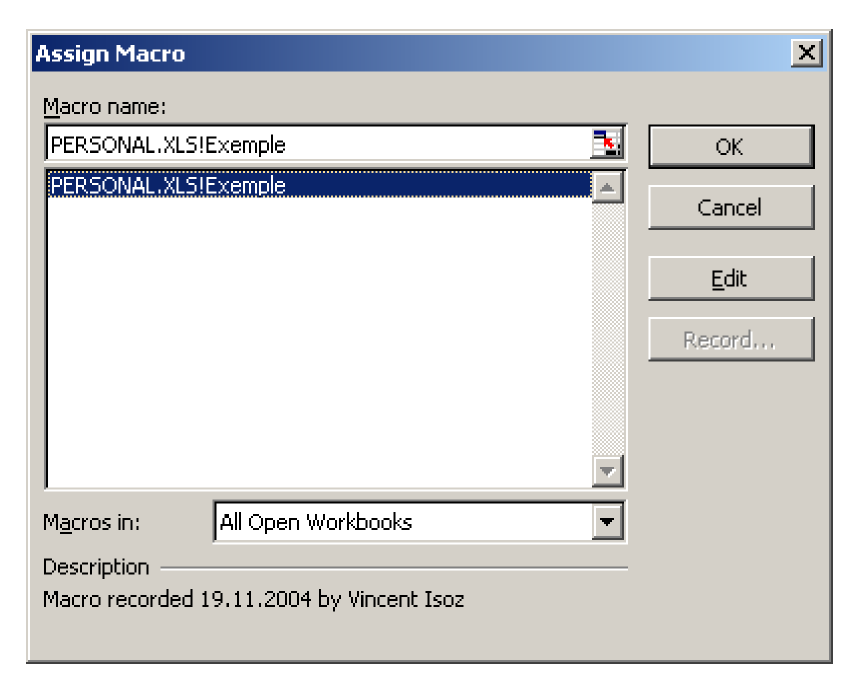


Dans MS Excel 2007 et ultérieur ces mêmes outils se trouvent dans l'onglet **Développeur**:

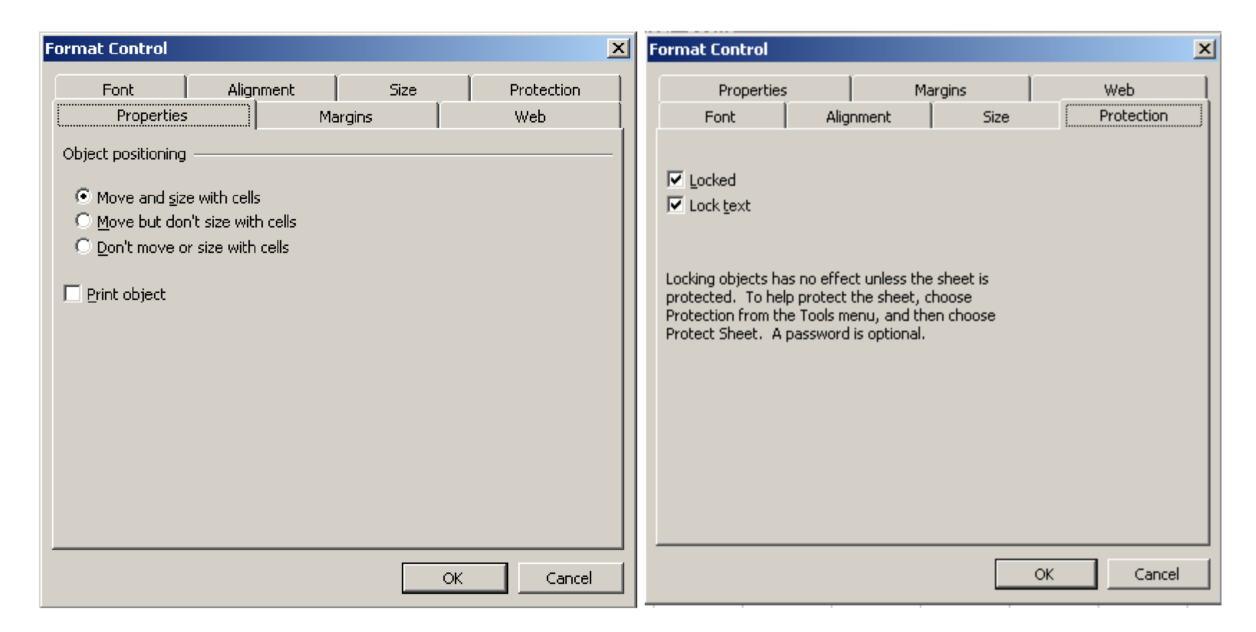


Voyons comment utiliser les boutons, les boutons d'options (les cases à cocher se basent sur le même principe) et les listes déroulantes:

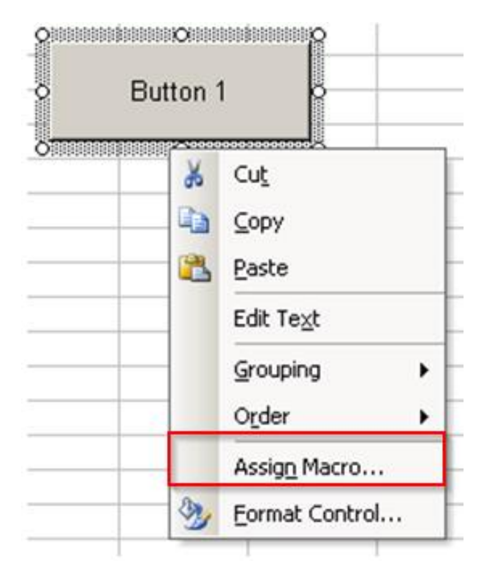
1. Pour créer un bouton sur une feuille qui permet d'exécuter une macro cliquez sur le bouton de la barre d'outils formulaires et dessinez en un. Au moment même où vous lâcherez la souris, la fenêtre suivante apparaîtra:



Vous demandant quelle macro vous souhaitez affecter à ce bouton. Vous faites simplement votre choix et cliquez ensuite sur **OK**. Vous pouvez ensuite modifier le texte du bouton, sa taille son formatage et ses propriétés ainsi qu'au besoin le supprimer (les plus importantes y relatives sont indiquées ci-dessous):



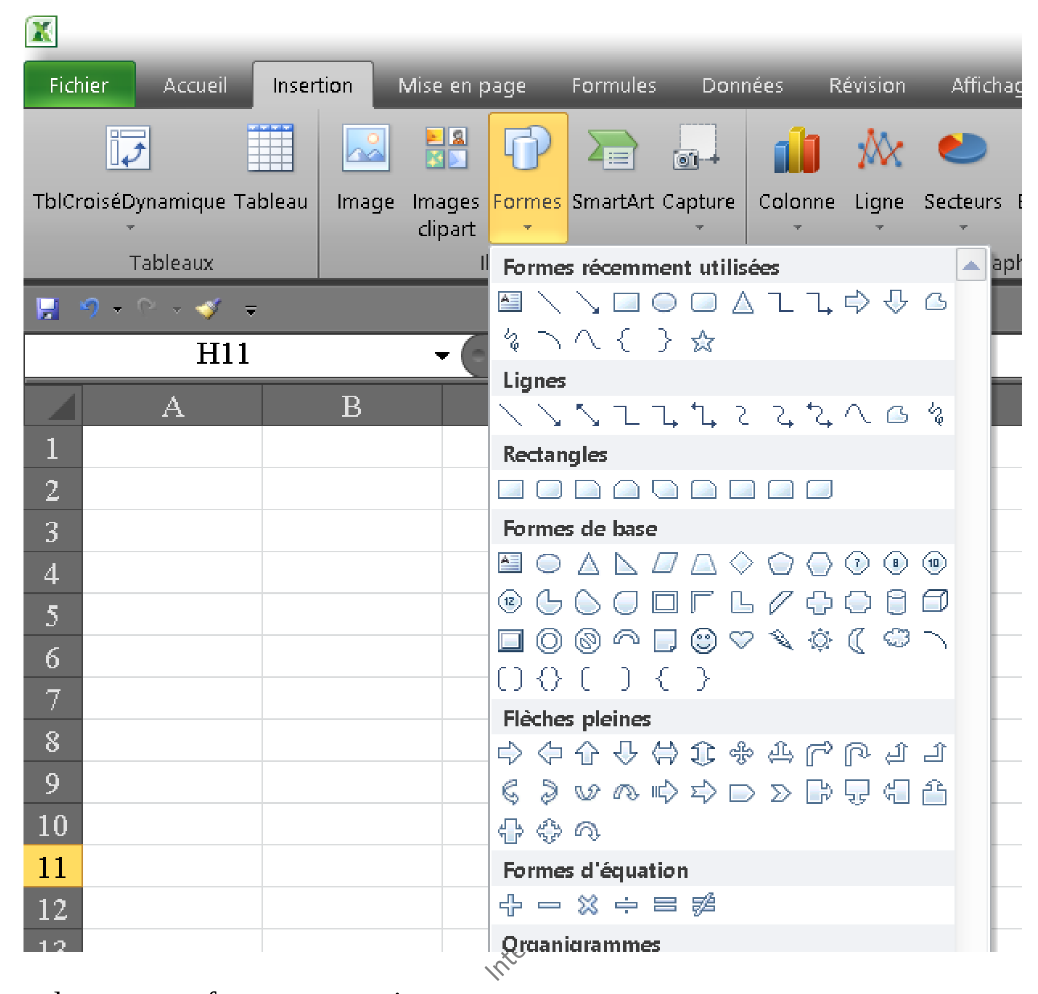
Pour affecter une autre macro au besoin sur ce bouton il suffit de faire un clic droit dessus et choisir **Affecter une macro** tel que indiqué ci-dessous:



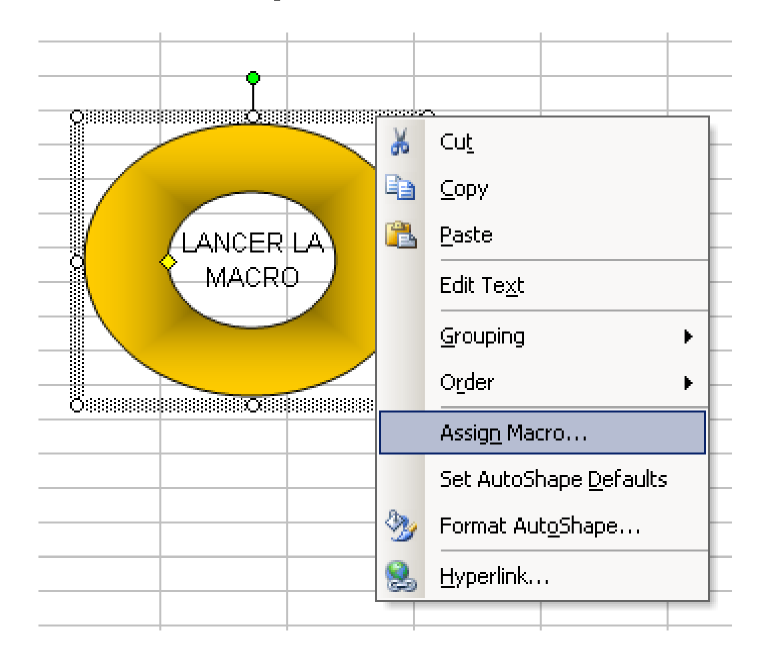
Au fait, la procédure est la même si vous souhaitez affecter des macros à des dessins ou des images (ce qui peut s'avérer esthétique) en utilisant la barre d'outils **Dessins** (vue au cours initiation) de MS Excel 2003 et antérieur:



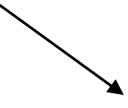
Sinon pour rappel dans les versions 2007 et ultérieur les formes géométrique se trouvent dans l'onglet **Insertion**:

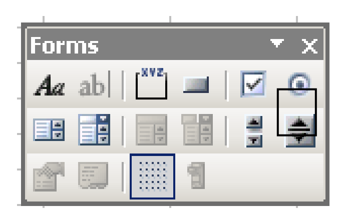


Par exemple avec une forme automatique:



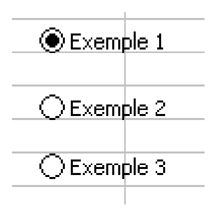
Voyons maintenant les boutons d'options:



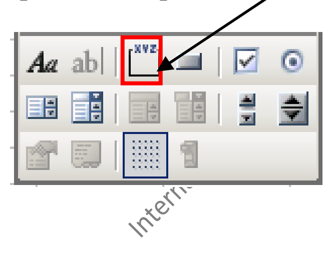


Pour les créer la méthode est la même que pour les boutons. Pour affecter une macro aussi !

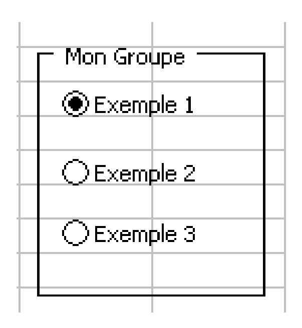
Ceci ayant été dit, si vous créez un groupe de boutons d'options:



Vous remarquerez sur la feuille qu'ils sont tous liés les uns aux autres (dans le sens que vous ne pouvez pas pour l'instant en activer plus de deux en même temps). Mais alors que faire si vous désirez faire plusieurs groupes de boutons d'options? Au fait, c'est simple ! Il suffit de les encadrer par une boîte de regroupement disponible ici:

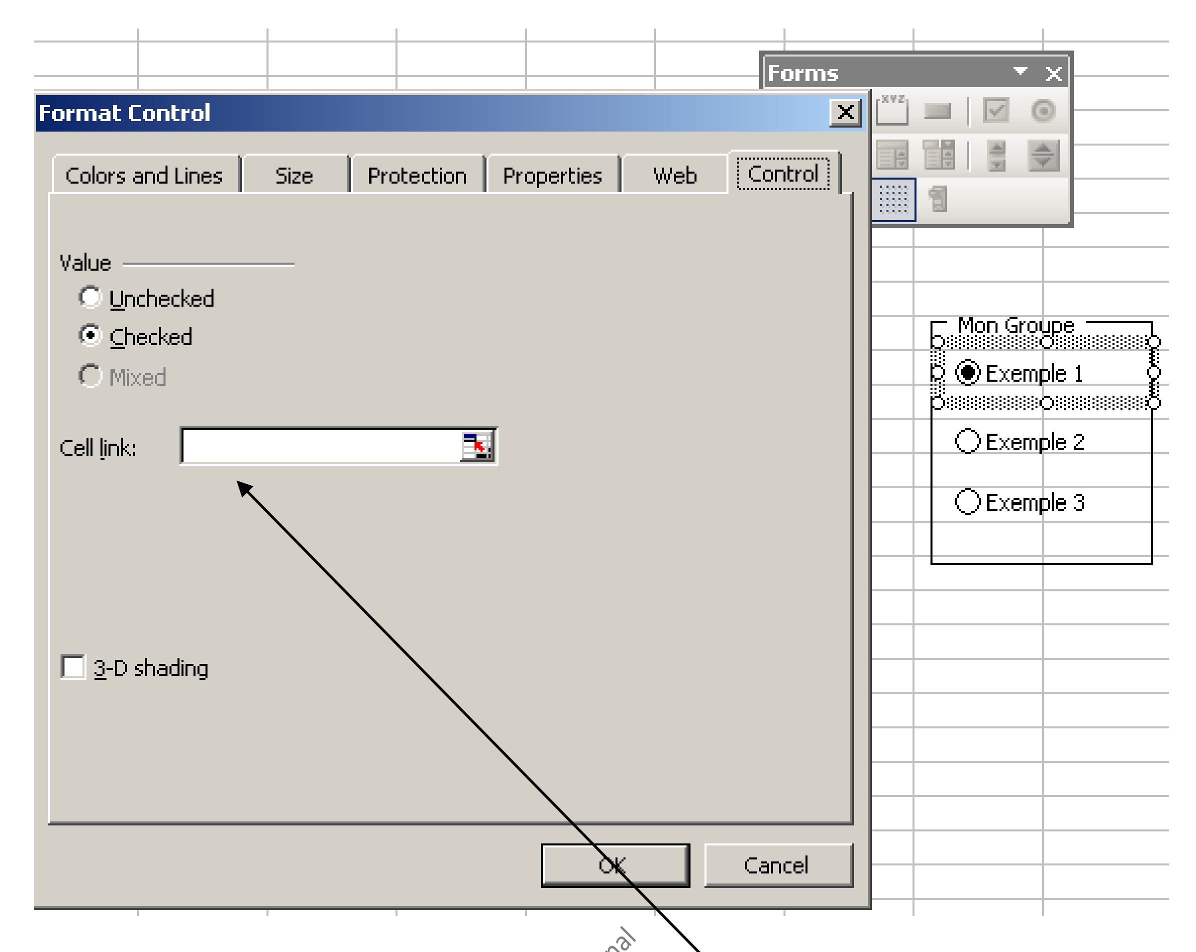


Ce qui donne:



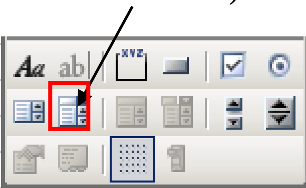
Ensuite, si vous souhaitez que tout se déplace ensemble (au cas où) il suffit de regrouper tous ces éléments comme s'il s'agissait de simples éléments de dessin (voir cours MS PowerPoint).

Mais quel est l'intérêt outre que l'on peut affecter une macro à chacun de ces butons radio? Eh bien tout simplement que l'on peut associer leur état à une cellule de manière à provoquer des calculs en cascades pour des usages ultérieur de macros ! Dès lors, comment faire cette liaison? Par un simple clic droit et accès aux propriétés d'un des boutons d'option du groupe tel que ceci apparaisse à l'écran:



Et ensuite spécifier la cellule désirée dans le champ **Cellule liée** ! Ainsi, si le premier bouton d'option est choisi, la valeur dans la cellule liée sera "1", "2" pour le deuxième, "3" pour le troisième, et ainsi de suite. Cette valeur peut ensuite être utilisée de manière très pertinente avec des fonctions SI, INDEX, RECHERCHEV qui seront elles-mêmes utilisées dans le cadre de l'exécution de macros automatiques.

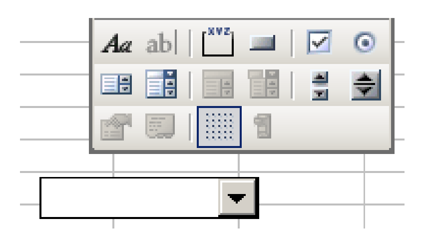
Voyons maintenant le dernier élément qui est certainement le plus utilisé. Il s'agit des listes déroulantes (auxquelles vous pouvez aussi affecter une macro en faisant un clic droit de la souris et qui s'exécutera après le choix de l'utilisateur):



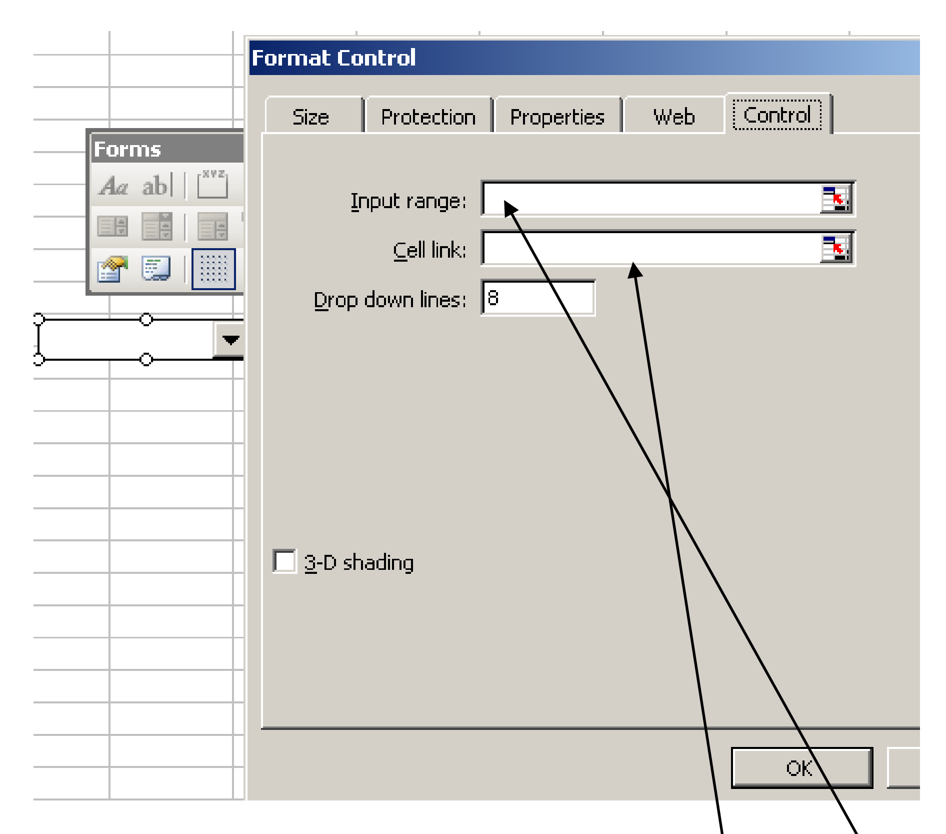
Avant il faut avoir quelque part une liste de données (qui peut provenir de n'importe où: une requête MS Access, d'un filtre élaboré, une liste statique, etc.…) telle que:



Ensuite, il faut dessiner une liste déroulante comme vous l'avez fait pour les boutons et les boutons d'option tel que:



Ensuite pour "peupler" la liste déroulante il suffit d'accéder à ses propriétés (bouton droit et choisir **Format du contrôle**):



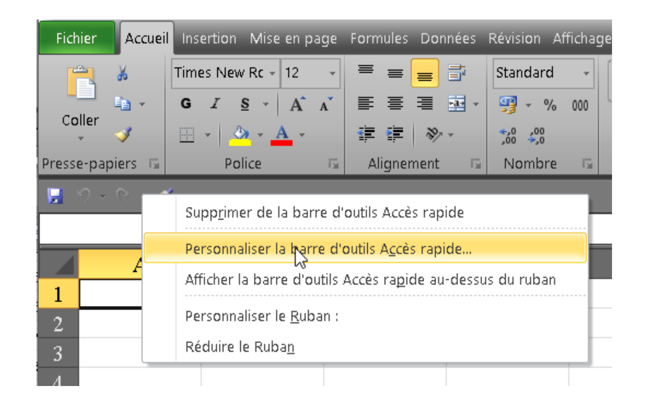
et définir la zone qui doit servir de remplissage pour la liste via le champ **Input Range**.

Remarque: De la même manière que pour les cases à cocher et boutons d'options vous pouvez lier une cellule aux choix effectué dans la liste via le champ **Cellule liée**. Ainsi, si le premier élément de la liste est choisi, la valeur dans la cellule liée sera "1", "2" pour le deuxième, "3" pour le troisième, et ainsi de suite. Cette valeur peut ensuite être utilisée de manière très pertinente avec des fonctions SI, INDEX, RECHERCHEV qui seront elles-mêmes utilisées dans le cadre de l'exécution de macros automatiques.

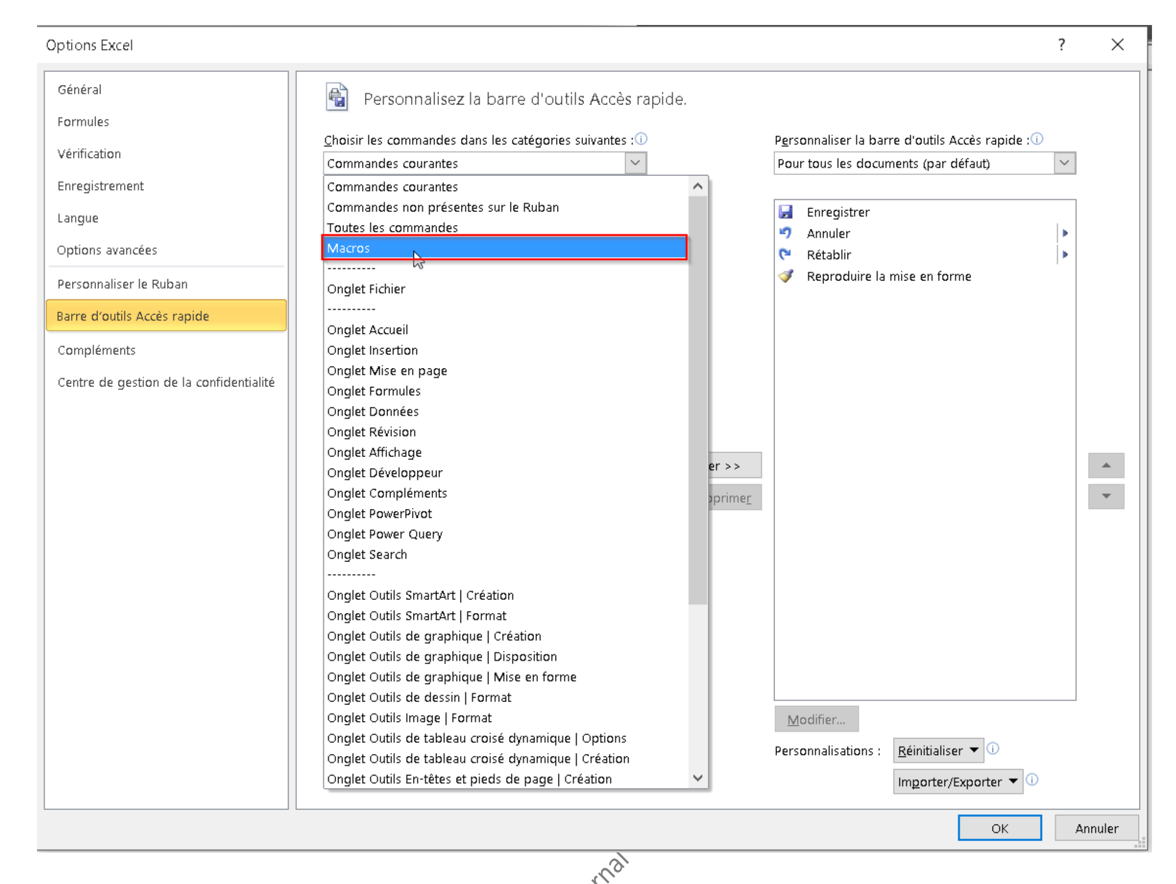
7.3.2 Barre d'accès rapide (MS Excel 2007 et ultérieur)

Comment faire pour avoir une macro qui apparaît dans la barre d'accès rapide chez tous vos collègues lorsqu'ils ouvrent un fichier MS Excel 2007 ou ultérieur contenant une de vos macros.

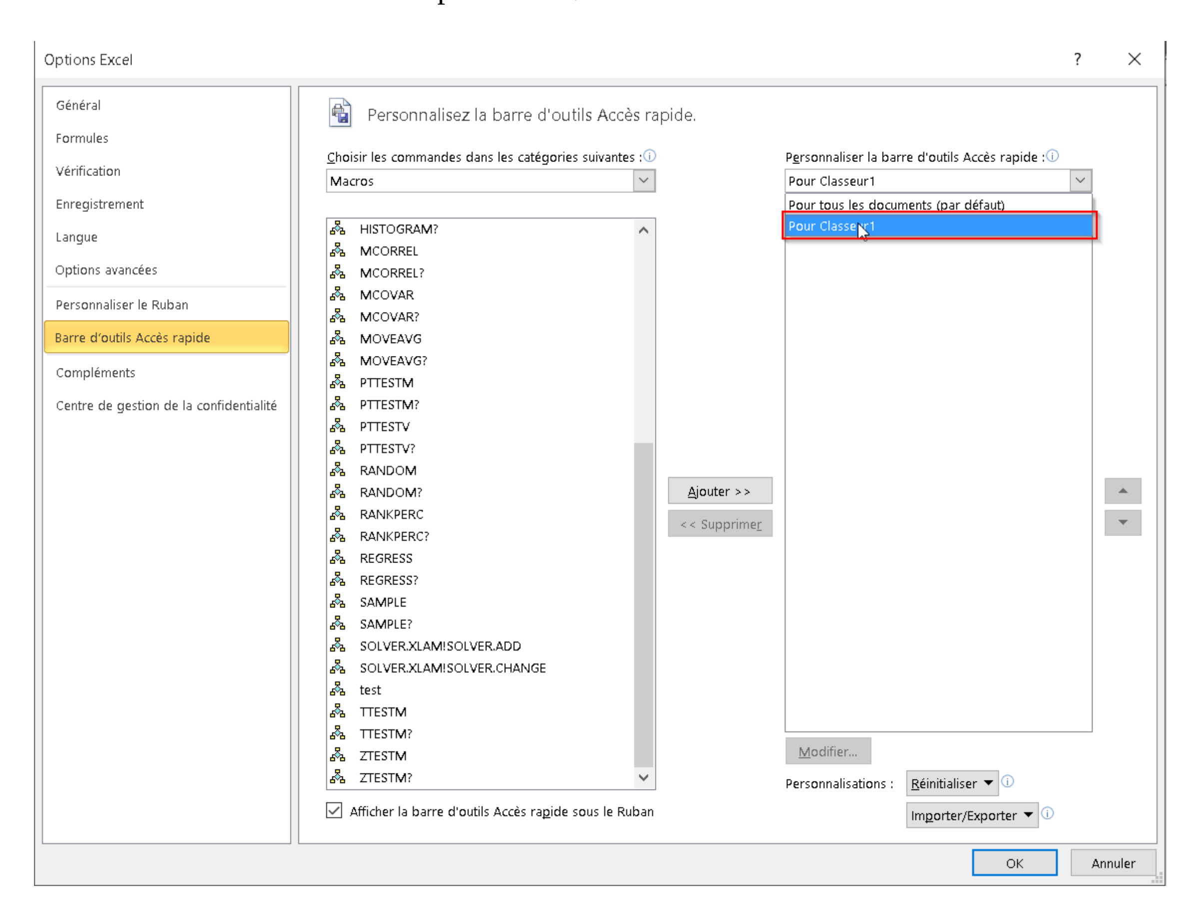
Considérons pour cela que vous avez créé par exemple une macro nommée *test* et faites un clic droit sur la barre d'accès rapide pour choisir l'option **Personnaliser la barre d'accès rapide**:



Ensuite dans la premère liste déroulante prenez **Macros**:



Dans la deuxième liste déroulante prenez le nom de votre fichier:



Une fois ceci fait sélectionnez votre macro à gauche et cliquez sur le bouton **Ajouter**:

Options Excel

Général
Formules
Vérification
Enregistrement
Langue
Options avancées
Personnaliser le Ruban
Barre d'outils Accès rapide
Compléments
Centre de gestion de la confidentialité

Personnalisez la barre d'outils Accès rapide.

Choisir les commandes dans les catégories suivantes :
Macros

HISTOGRAM?
MCORREL
MCORREL?
MCOVAR
MCOVAR?
MOVEAVG
MOVEAVG?
PTTESTM
PTTESTM?
PTTESTV
PTTESTV?
RANDOM
RANDOM?
RANKPERC
RANKPERC?
REGRESS
REGRESS?
SAMPLE
SAMPLE?
SOLVER.XLAM!SOLVER.ADD
SOLVER.XLAM!SOLVER.CHANGE
test
TTESTM
TTESTM?
ZTESTM
ZTESTM?

Afficher la barre d'outils Accès rapide sous le Ruban

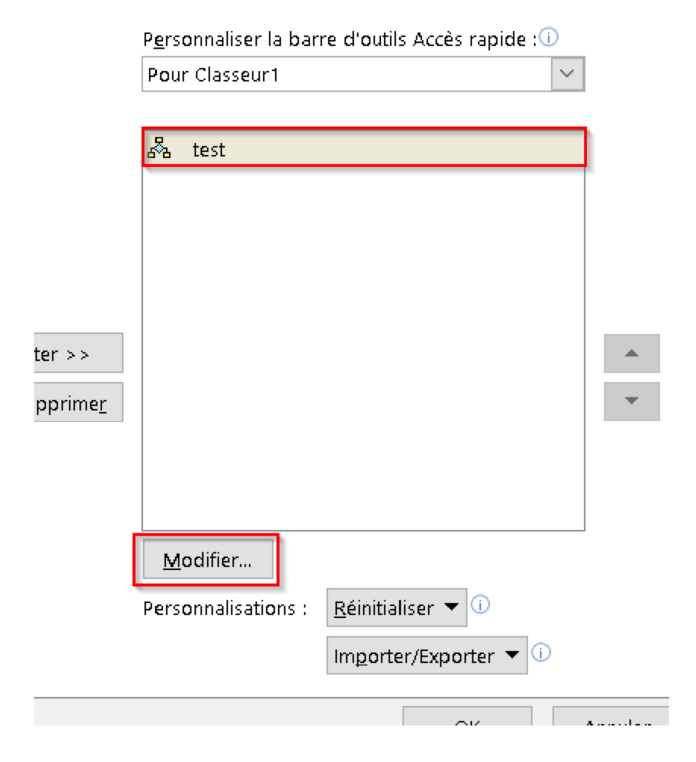
Ajouter >>
<< Supprimer

Personnaliser la barre d'outils Accès rapide :
Pour Classeur1

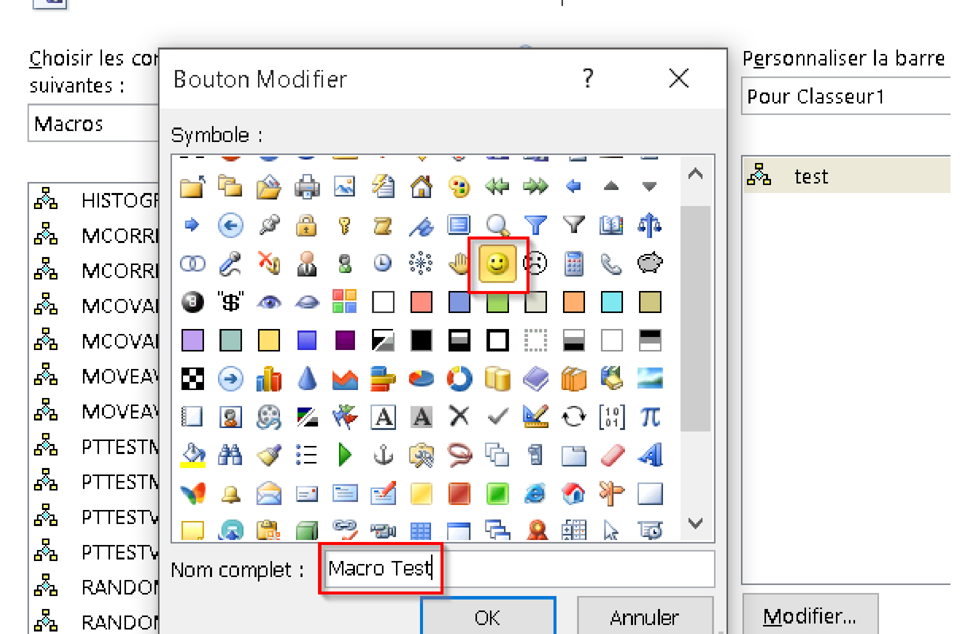
Modifier...
Personnalisations : Réinitialiser
Importer/Exporter

OK
Annuler

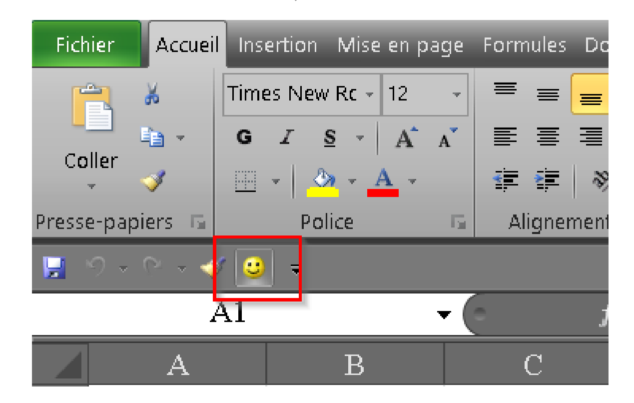
Une fois ceci fait, sélectionnez votre macro se trouvant maintenant à droite et cliquez sur le bouton **Modifier**:



Vous avez alors la possibilité de choisir une icône et de changer la légende de la macro:



Si vous validez deux fois par **OK** vous avez maintenant un bouton qui suivre votre fichier MS Excel sur tous les ordinateurs de vos collègues:



7.3.3 Barres d'outils et menus (MS Excel 2003 et antérieur)

Les barres d'outils et les éléments de menu sont normalement créés dans le cadre de l'utilisation de macros personnelles mais elles sont surtout utilisées dans la cadre de la création de macros complémentaires.

Dans un premier temps, nous considérons qu'il est connu du participant comme créer des barres d'outils, menus et comment affecter une macro à ces différents éléments pour les raisons suivantes:

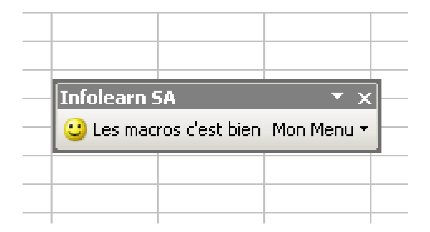
1. C'est super simple

2. Cela est normalement vu pendant le cours MS Excel avancé (cours qui précède le cours Macro et VBA)

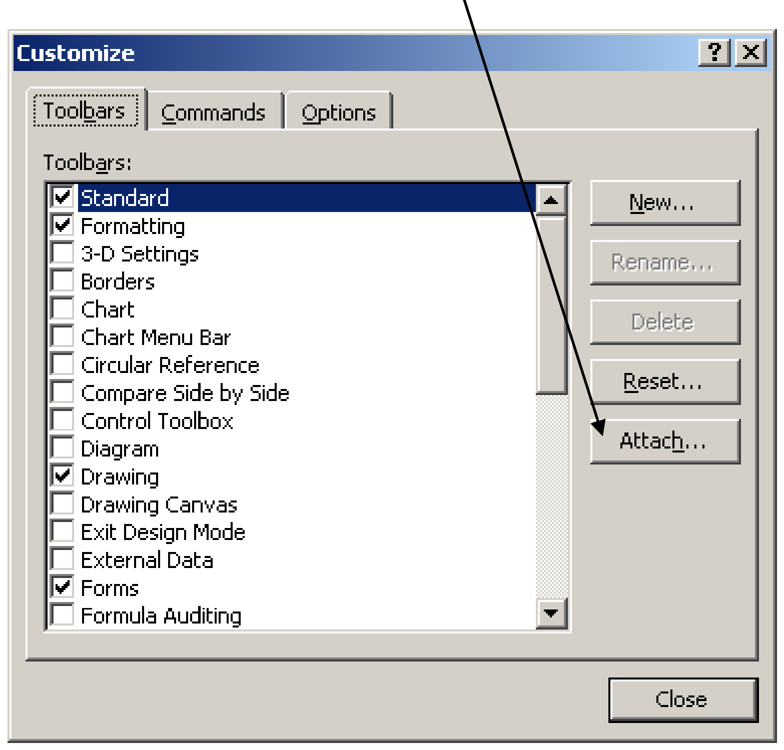
3. Vous avez la procédure normalement dans un peu près n'importe quel bouquin avancé traitant de MS Excel.

Cependant il y normalement un élément qui n'est pas vu dans le cours avancé: c'est comment lier une barre d'outils à un classeur de telle manière que lorsque celui-ci est envoyé à un collègue la barre d'outils personnalisée s'installe automatiquement sur son poste.

Comment faire cela? Eh bien imaginons la barre personnelle suivante:



Nous souhaiterions l'attacher à notre classeur (n'oubliez pas que dès que quelqu'un ouvrira le classeur en question, cette barre d'outils sera installée dans son système). Pour ce faire il suffit d'aller dans la personnalisation des barres d'outils et:



Après la suite est évidente. Il faut cependant prendre garde à une chose: lorsque vous faites une mise à jour de votre barre d'outils, il faut re-attacher celle-ci !!!!!!!!!!!!!

7.3.4 Ruban et onglets (MS Excel 2007 et ultérieur)

Le domaine des rubans est très vaste, j'ajouterai des informations au fur et à mesure sur le sujet.

Signalons tout de suite qu'il n'est pas possible à ma connaissance dans Excel 2010 de modifier le ruban avec du V.B.A. contrairement à MS Access.

7.3.4.1 Masquer totalement le ruban

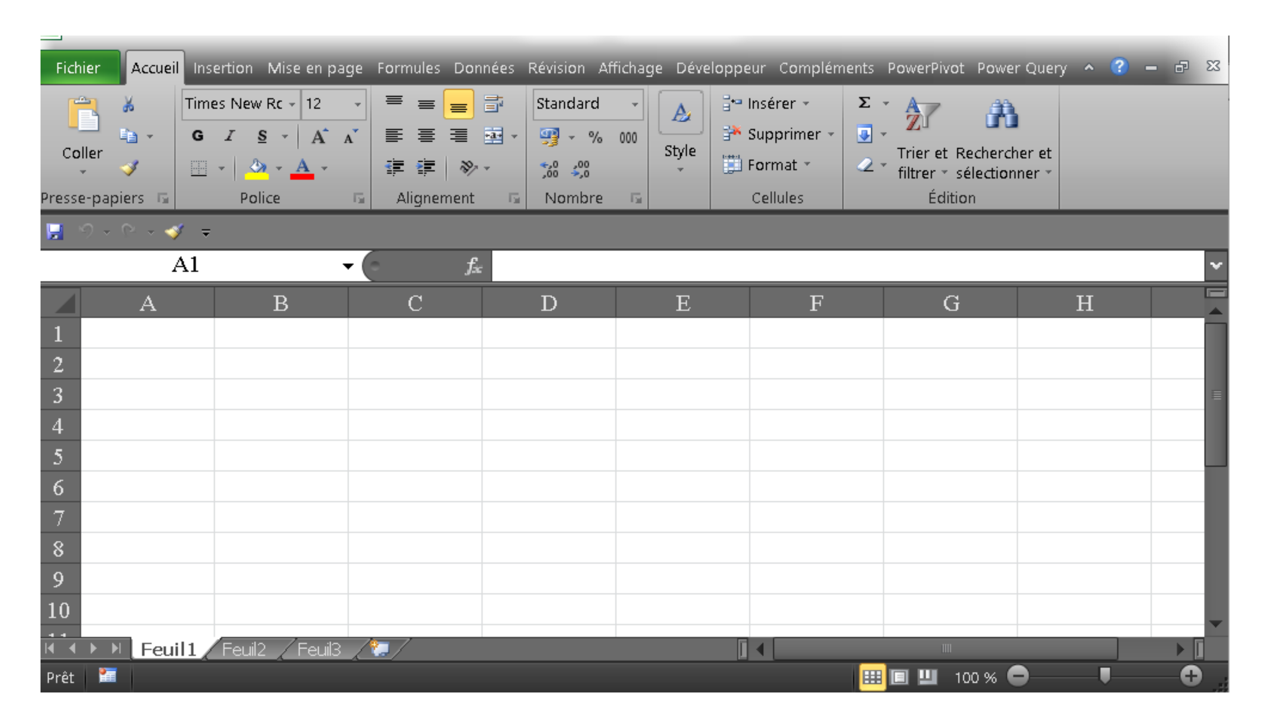
Commençons déjà par la commande suivante dans n'importe quel module, procédure ou fonction qui est très utile:

```
Application.ExecuteExcel4Macro "SHOW.TOOLBAR(""Ribbon"",False)"
```

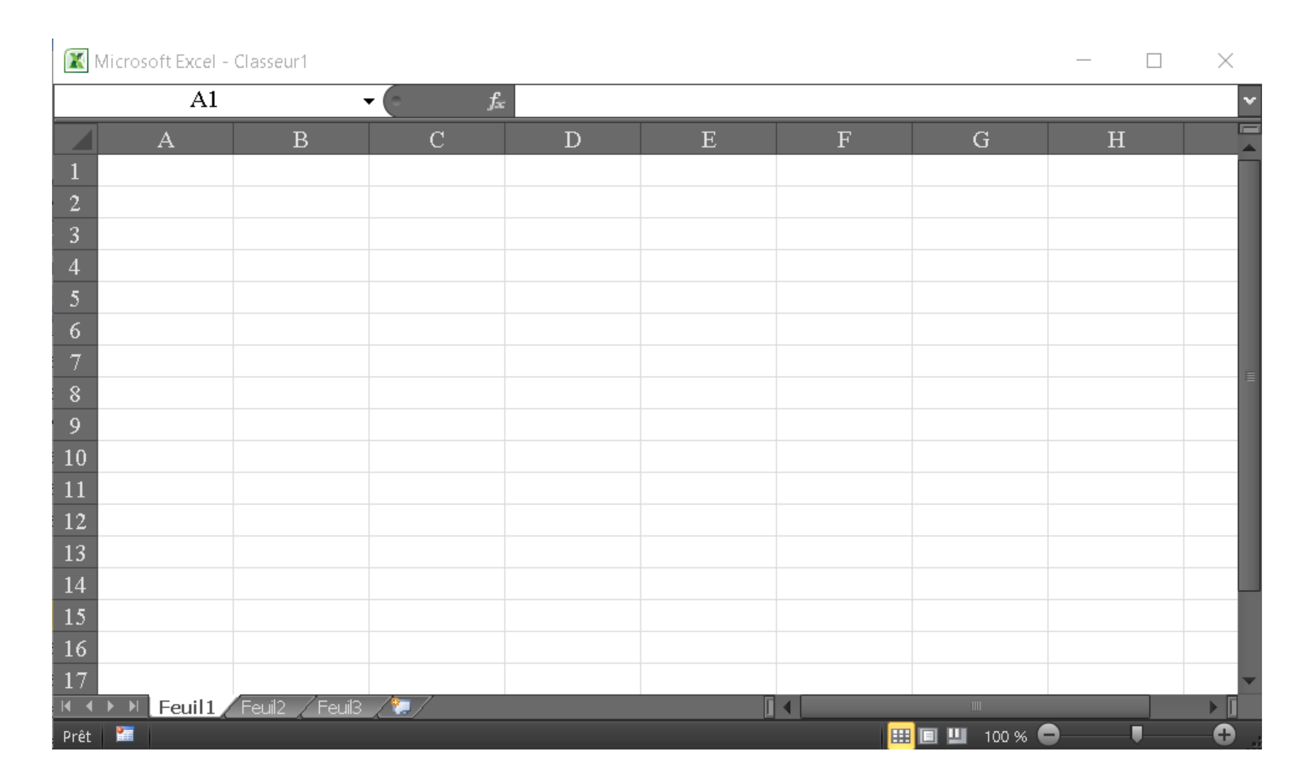
elle masque totalement le ruban ou le remet:

```
Application.ExecuteExcel4Macro "SHOW.TOOLBAR(""Ribbon"",True)"
```

Effectivement! Avant:

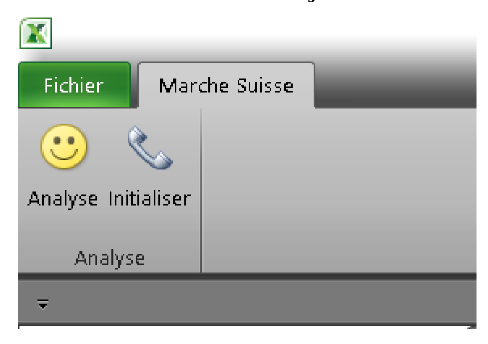


Après:



7.3.4.2 Personnalisation du ruban

Notre objectif ici va être très simple! C'est d'obtenir en gros le résultat suivant (on peut faire beaucoup plus compliqué et tordu mais cela fait l'objet d'une formation sur le RibbonX):

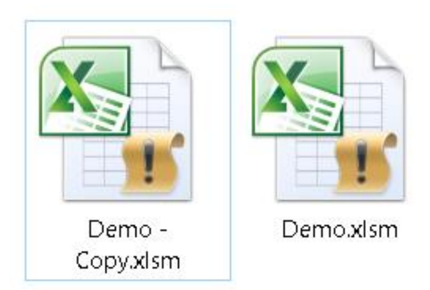


Pour ce faire voici la procédure.

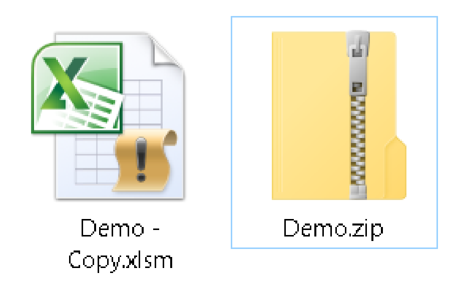
1. Votre fichier MS Excel contenant vos macros doit être soit au format *.xlsm, soit *.xltm, soit *.xlsam mais ni *.xls ou *.xlsb:



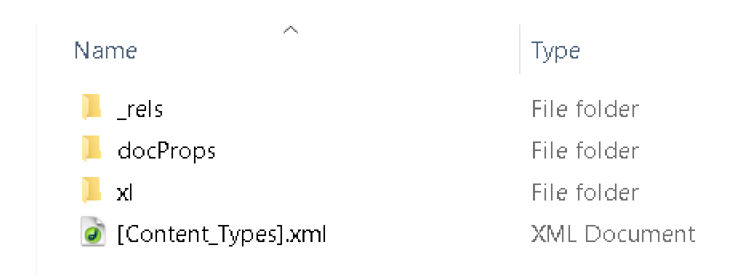
2. Ensuite il faut en faire faire une copie et garder la copie au chaud en cas d'erreur de manipulation fatale (on est jamais assez prudent!):



3. Ensuite changez l'extension du fichier en *.zip:

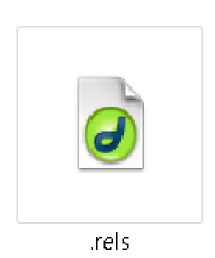


4. Entrez dans le *.zip pour y voir la structure suivante:



Name	Type
_rels	File folder
docProps	File folder
xl	File folder
[Content_Types].xml	XML Document

5. Entrez dans le dossier **_rels**:



6. Ouvrez le fichier **.rels** (il est possible que vous deviez le sortir du fichier *.zip pour l'éditer) et assurez-vous d'y rajouter ce qu'il faut pour avoir:

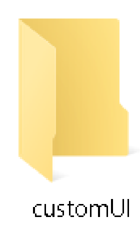
```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Relationships xmlns="http://schemas.openxmlformats.org/package/2006/relationships">
<Relationship Id="rId3" Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/extended-properties" Target="docProps/app.xml"/>
<Relationship Id="customUIRelID" Type="http://schemas.microsoft.com/office/2006/relationships/ui/extensibility" Target="customUI/customUI.xml"/>
<Relationship Id="rId2" Type="http://schemas.openxmlformats.org/package/2006/relationships/metadata/core-properties" Target="docProps/core.xml"/>
<Relationship Id="rId1" Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/officeDocument" Target="xl/workbook.xml"/>
</Relationships>
```

Soit pour ceux qui veulent copier/coller:

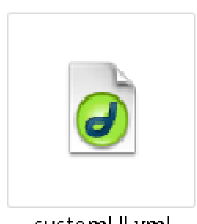
```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
```

```
   <Relationships
xmlns="http://schemas.openxmlformats.org/package/2006/relationships">
      <Relationship Id="rId3"
Type="http://schemas.openxmlformats.org/officeDocument/2006/relations
hips/extended-properties" Target="docProps/app.xml"/>
      <Relationship Id="customUIRelID"
Type="http://schemas.microsoft.com/office/2006/relationships/ui/exten
sibility" Target="customUI/customUI.xml"/>
      <Relationship Id="rId2"
Type="http://schemas.openxmlformats.org/package/2006/relationships/me
tadata/core-properties" Target="docProps/core.xml"/>
      <Relationship Id="rId1"
Type="http://schemas.openxmlformats.org/officeDocument/2006/relations
hips/officeDocument" Target="xl/workbook.xml"/>
</Relationships>
```

7. Maintenant, dans le dossier *.zip créez un dossier **customUI**:



8. Créez à l'intérieur de ce dossier un fichier customUI.xml:



avec dedans:

```
<customUI xmlns="http://schemas.microsoft.com/office/2006/01/customui">
  <ribbon startFromScratch="true">
    <tabs>
      <tab id="CustomTab" label="Marche Suisse">
        <group id="SimpleControls" label="Analyse">
          <button id="Button1" imageMso="HappyFace" size="large" label="Analyse" onAction="ThisWorkbook.MSAnalyse" />
          <button id="Button2" imageMso="AutoDial" size="large" label="Initialiser" onAction="ThisWorkbook.MSInit" />
        </group>
      </tab>
    </tabs>
  </ribbon>
</customUI>
```

Soit pour ceux qui veulent copier/coller:

```
<customUI
xmlns="http://schemas.microsoft.com/office/2006/01/customui">
  <ribbon startFromScratch="true">
    <tabs>
      <tab id="CustomTab" label="Marche Suisse">
        <group id="SimpleControls" label="Analyse">
          <button id="Button1" imageMso="HappyFace" size="large"
```

```
label="Analyse" onAction="ThisWorkbook.MSAnalyse" />
          <button id="Button2" image"AutoDial" size="large"
label="Initialiser" onAction="ThisWorkbook.MSInit" />
        </group>
      </tab>
    </tabs>
  </ribbon>
</customUI>
```

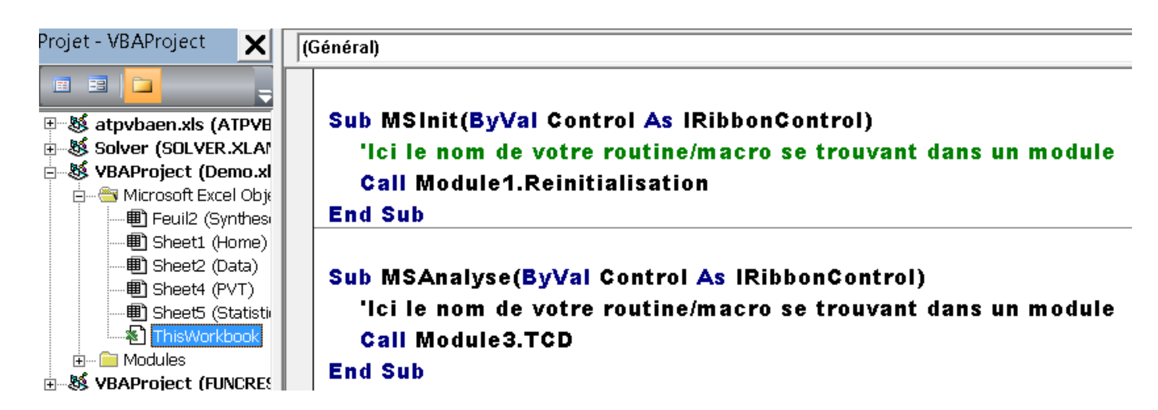
Vous avez alors:



9. Ensuite il faut renommer le fichier *.zip dans son extension d'origine (*.xlsm, *.xltm ou *.xlam):



10. Enfin, il suffit de rajouter le code V.B.A. suivant dans l'objet **ThisWorkbook**:



Pour ceux qui veulent plus d'exemple et approfondir le sujet, vous pouvez vous référer à:

https://silkyroad.developpez.com/excel/ruban/

7.3.4.3 Faire apparaître/disparaître un onglet du ruben en VBA

Une question rare mais récurrente dans les entreprises et la possibilité de faire apparaître ou disparaître un un onglet du ruban à partir d'un événement VBA (clic sur un bouton, changement de feuille, ou autre…).

La méthode est très loin d'être simple… et voici donc un exemple avec le code à copier/coller (qui fonctionne depuis Microsoft Excel 2007).

First we create a module with the following code:

```
Option Explicit

Dim Rib As IRibbonUI
Public MyTag As String

'Callback for customUI.onLoad
Sub RibbonOnLoad(ribbon As IRibbonUI)
    Set Rib = ribbon
End Sub

Sub GetVisible(control As IRibbonControl, ByRef visible)
    If MyTag = "show" Then
        visible = True
    Else
        If control.Tag Like MyTag Then
            visible = True
        Else
            visible = False
        End If
    End If
End Sub

Sub RefreshRibbon(Tag As String)
    MyTag = Tag
    If Rib Is Nothing Then
        MsgBox "Error, Save/Restart your workbook"
    Else
        Rib.Invalidate
    End If
End Sub

'Note: Do not change the code above

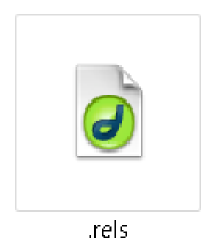

'**************************************************************************
'Examples to show only the Tab you want with getVisible and tag in the
RibbonX.
'**************************************************************************

Sub DisplayRibbonTab()
   'Show only the Tab, Group or Control with the Tag "Swiss"
   'this is the routine you can associate to the button or event you want!
    Call RefreshRibbon(Tag:="Swiss")
End Sub

'Note: in this example every macro above will show you the custom tab.
'If you add more custom tabs this will be different
```

```
Sub HideEveryTab()
    'Hide every Tab, Group or Control(we use Tag:="")
     Call RefreshRibbon(Tag:="")
End Sub
```

Ceci fait, comme pour l'exemple plus haut, entrez dans le dossier **_rels**:



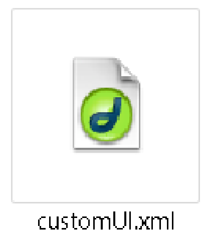
Ouvrez le fichier **.rels** (il est possible que vous deviez le sortir du fichier *.zip pour l'éditer) et assurez-vous d'y rajouter ce qu'il faut pour avoir (c'est donc parfaitement identique à l'exemple vu plus haut!):

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Relationships xmlns="http://schemas.openxmlformats.org/package/2006/relationships">
<Relationship Id="rId3" Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/extended-properties" Target="docProps/app.xml"/>
<Relationship Id="customUIRelID" Type="http://schemas.microsoft.com/office/2006/relationships/ui/extensibility" Target="customUI/customUI.xml"/>
<Relationship Id="rId2" Type="http://schemas.openxmlformats.org/package/2006/relationships/metadata/core-properties" Target="docProps/core.xml"/>
<Relationship Id="rId1" Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/officeDocument" Target="xl/workbook.xml"/>
</Relationships>
```

Soit pour ceux qui veulent copier/coller:


```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
   <Relationships
xmlns="http://schemas.openxmlformats.org/package/2006/relationships">
      <Relationship Id="rId3"
Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/e
xtended-properties" Target="docProps/app.xml"/>
      <Relationship Id="customUIRelID"
Type="http://schemas.microsoft.com/office/2006/relationships/ui/extensibili
ty" Target="customUI/customUI.xml"/>
      <Relationship Id="rId2"
Type="http://schemas.openxmlformats.org/package/2006/relationships/metadata
/core-properties" Target="docProps/core.xml"/>
      <Relationship Id="rId1"
Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/o
fficeDocument" Target="xl/workbook.xml"/>
</Relationships>
```

ensuite dans le fichier customUI.xml:



Nous mettons:

```
<customUI onLoad="RibbonOnLoad" xmlns="http://schemas.microsoft.com/office/2006/01/customui">
  <ribbon startFromScratch="true">
    <tabs>
      <tab id="CustomTab1" label="Marche Suisse" getVisible="GetVisible" tag="Swiss" >
        <group id="SimpleControls1" label="Analyse">
          <button id="Button1" imageMso="HappyFace" size="large" label="Analyse" onAction="ThisWorkbook.MSAnalyse" />
          <button id="Button2" imageMso="AutoDial" size="large" label="Initialiser" onAction="ThisWorkbook.MSInit" />
        </group>
      </tab>
      <tab id="CustomTab2" label="Marche Francais" getVisible="GetVisible" tag="France" >
        <group id="SimpleControls2" label="Analyse">
          <button id="Button3" imageMso="HappyFace" size="large" label="Analyse" onAction="ThisWorkbook.MSAnalyse" />
          <button id="Button4" imageMso="AutoDial" size="large" label="Initialiser" onAction="ThisWorkbook.MSInit" />
        </group>
      </tab>
    </tabs>
  </ribbon>
</customUI>
```

où nous avons mis dans des rectangles rouges les éléments importants spécifique à note exemple!

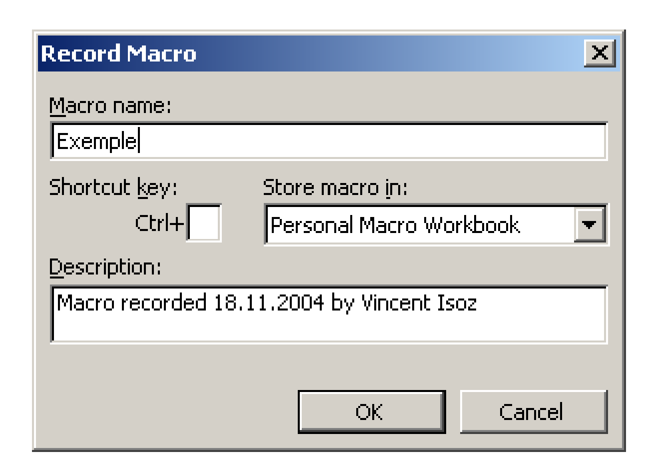
Soit pour ceux qui veulent copier/coller:

```
<customUI onLoad="RibbonOnLoad"
xmlns="http://schemas.microsoft.com/office/2006/01/customui">
  <ribbon startFromScratch="true">
    <tabs>
      <tab id="CustomTab1" label="Marche Suisse" getVisible="GetVisible"
tag="Swiss" >
        <group id="SimpleControls1" label="Analyse">
          <button id="Button1" imageMso="HappyFace" size="large"
label="Analyse" onAction="ThisWorkbook.MSAnalyse" />
          <button id="Button2" imageMso="AutoDial" size="large"
label="Initialiser" onAction="ThisWorkbook.MSInit" />
        </group>
      </tab>
        <tab id="CustomTab2" label="Marche Francais"
getVisible="GetVisible" tag="France" >
        <group id="SimpleControls2" label="Analyse">
          <button id="Button3" imageMso="HappyFace" size="large"
label="Analyse" onAction="ThisWorkbook.MSAnalyse" />
          <button id="Button4" imageMso="AutoDial" size="large"
label="Initialiser" onAction="ThisWorkbook.MSInit" />
        </group>
      </tab>
    </tabs>
  </ribbon>
</customUI>
```


La suite étant parfaitement identique au premier exemple concernant comment *Masquer le ruban.*

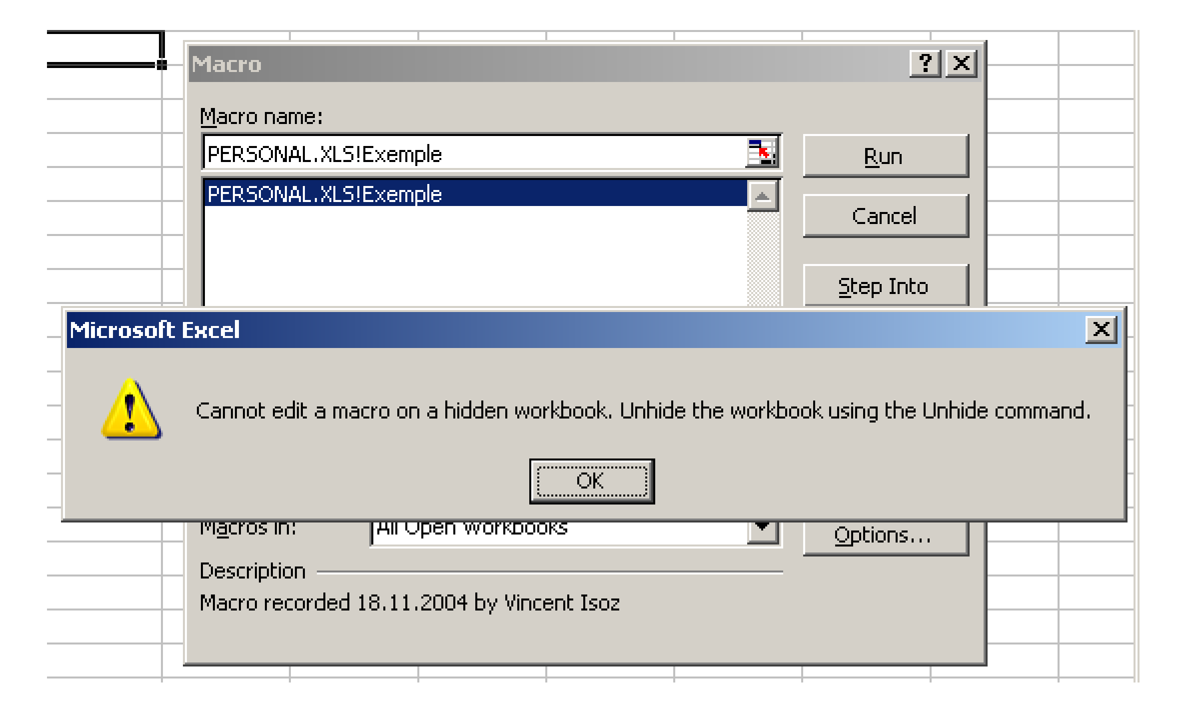
7.4 Macros personnelles

Nous avons vu tout à l'heure comme enregistrer une macro dans un classeur. Cependant, il peut tout à fait arriver que vous ayez besoin d'une macro enregistrée qui fonctionne dans tous les classeurs de VOTRE ordinateur. Dès lors, il faudra enregistrer votre macro (en absolu ou relative ou un mélange) dans le **Classeur de macros personnelles** comme indiqué ci-dessous:



Dès lors, lorsque que vous créerez une barre d'outils ou un onglet avec un bouton d'accès (nous verrons comment faire cela un peu plus loin) à cette même macro, celle-ci s'exécutera sur tout ancien, présent ou futur classeur MS Excel.

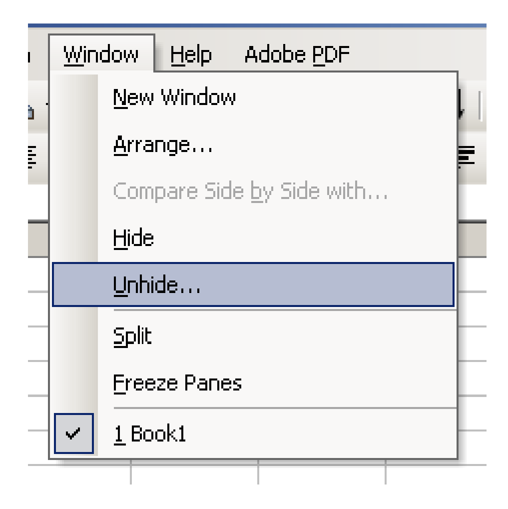
Hors l'étendue de fonctionnement une autre différence des macros personnelles est qu'elles ne peuvent pas être supprimées à l'aide de la boîte de dialogue disponible à l'aide de **Alt+F8**. Effectivement:



Dès lors, deux possibilités s'offrent à vous (préférez la seconde à la première):

1. Activer l'affichage du *PERSONAL.xls* puis supprimer la macro (parce que par défaut ce fichier étant masqué est tel que nous ne pouvons pas y supprimer les macros en passant par cette fenêtre).

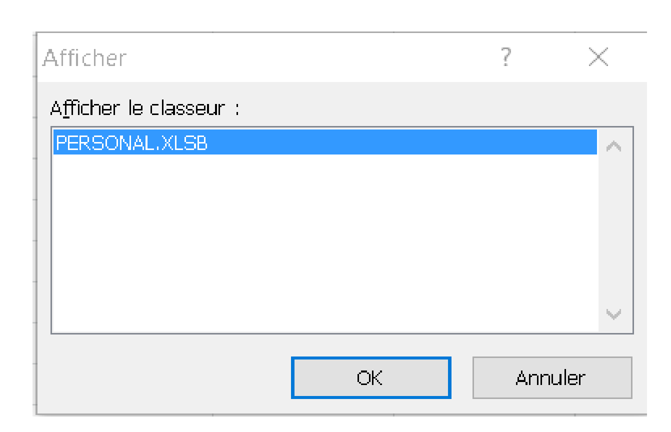
   Dans MS Excel 2003 et antérieur:



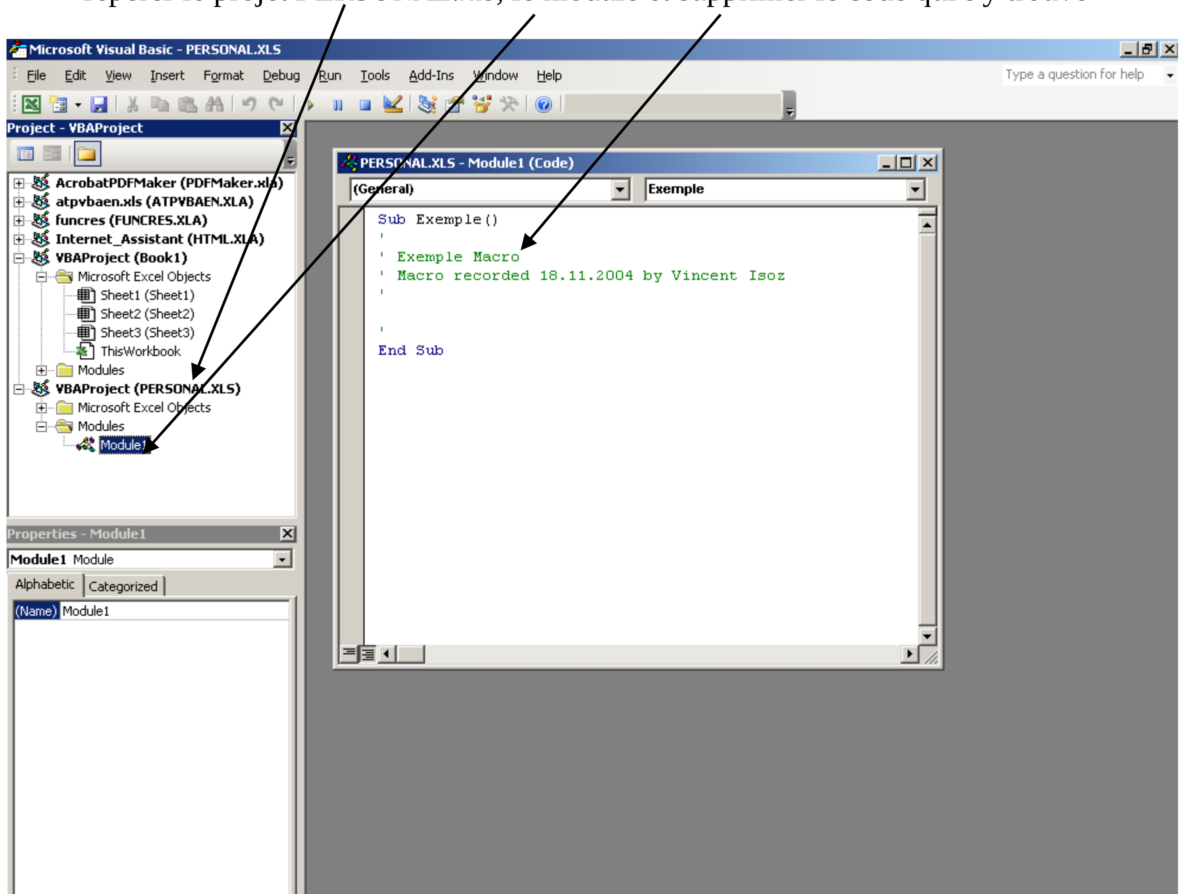
   Dans MS Excel 2007 et ultérieur:



   Il apparaît alors la fenêtre habituelle des fichiers masqués:



2. Aller dans le VBAE (Visual Basic Application Editor) par le raccourci **Alt+F11**, repérer le projet *PERSONAL.xls*, le module et supprimer le code qui s'y trouve



L'endroit où est stocké le fichier PERSONAL.XLS dépend parfois des entreprises. Le mieux est simplement d'afficher le fichier dans MS Excel et d'afficher ses propriétés. Sinon on peut considérer que la majorité du temps, il est dans le chemin:

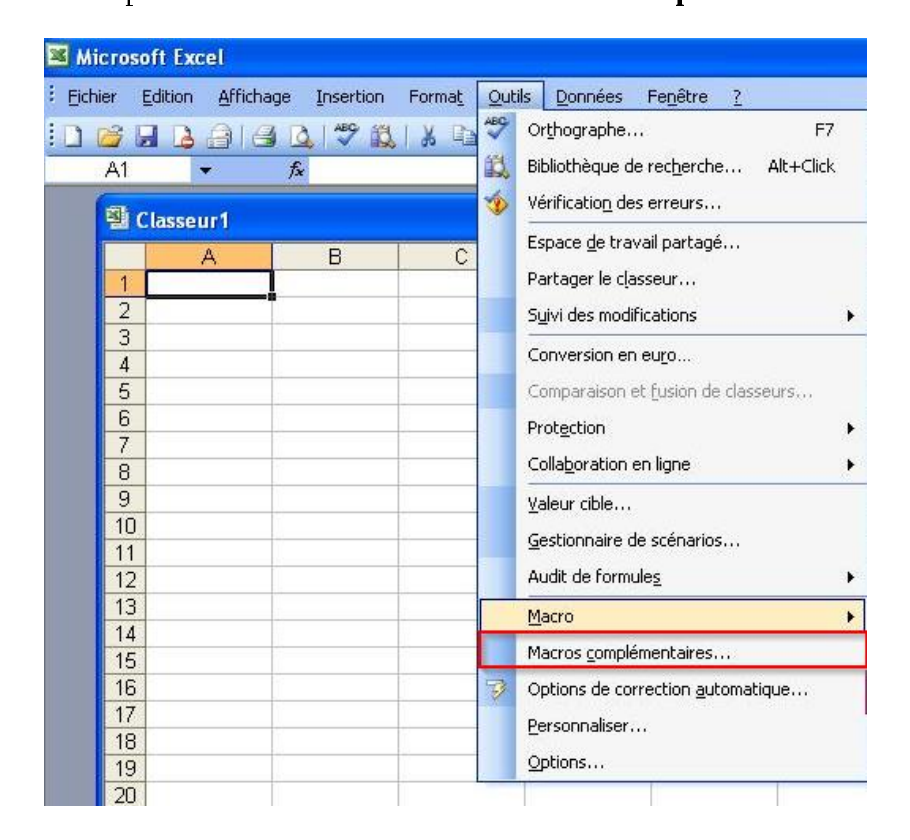
C:\Documents and Settings\'Profil'\Application Data\Microsoft\Excel\XLSTART

ou suivant les versions de Microsoft Windows:

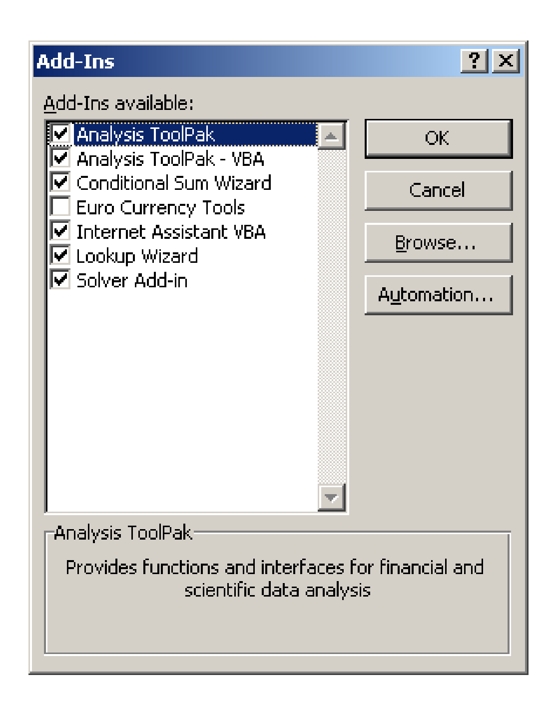
C:\Users\'Profil'\Application Data\Roaming\Microsoft\Excel\XLSTART

7.5 Macros complémentaires

C'est l'outil certainement le plus puissant de MS Excel puisqu'il a la possibilité de tout faire selon une stratégie optimale dans le cadre d'une entreprise que ce soit pour distribuer des programmes ou des algorithmes mathématiques. Par ailleurs, c'est la méthode utilisée par les professionnels des maisons d'éditions de logiciels pour créer des Plug-Ins dans MS Excel 2003 et antérieurs disponibles dans le menu **Outils/Macro Complémentaires**:



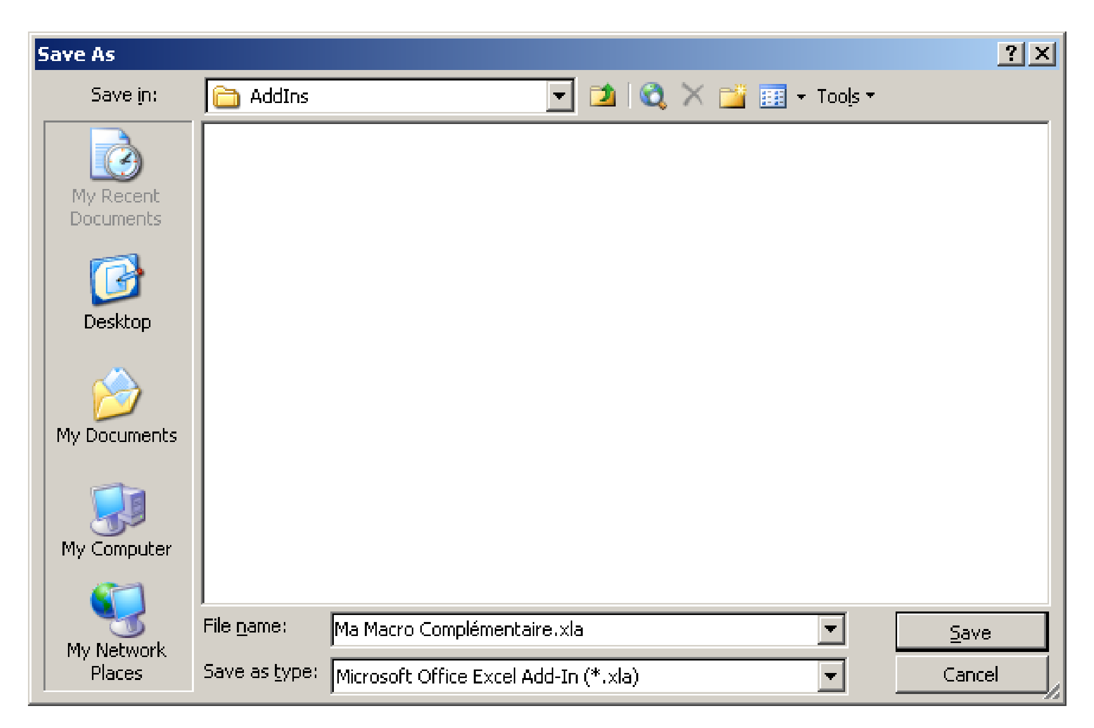
Et il apparaît alors:



Et dans MS Excel 2007 et utlérieur:

La procédure pour créer une macro complémentaire est la suivante:

1. Créer et enregistrer un nouveau classeur
2. Supprimer les feuilles inutiles
3. Enregistrer la macro comme à l'habitude
4. Créer une barre d'outils (MS Excel 2003 ou antérieur) ou un onglet (MS Excel 2007 ou utlérieur) avec les boutons nécessaires à l'exécution de la ou les macros (attention au nom de la barre en ce qui concernera les mises à jour de votre add-in)
5. Attacher la barre d'outils au classeur en cours pour MS Excel 2003 ou antérieur.
6. Enregistrer ensuite le classeur ensuite tant que macro complémentaire. Pour cela, il faut aller dans **Fichier/Enregistrer** Sous et dans **Type de fichier** choisir dans la liste **Macro complémentaire Microsoft Excel**:



Nous pouvons observer que les macros complémentaires sont des fichiers *.xla pour MS Excel 2003 et antérieure et *.xlam pour MS Excel 2007 et ultérieur. XLA étant l'abréviation de Excel Add-In (c'est cela que vous pouvez télécharger sur certains sites Internet traitant de MS Excel !!!). Un fichier add-in ressemble à:



Remarque: Si vous souhaitez qu'une macro complémentaire soit automatiquement installée sur le logiciel MS Excel d'un utilisateur, il suffit pour cela de la déployer dans le logiciel

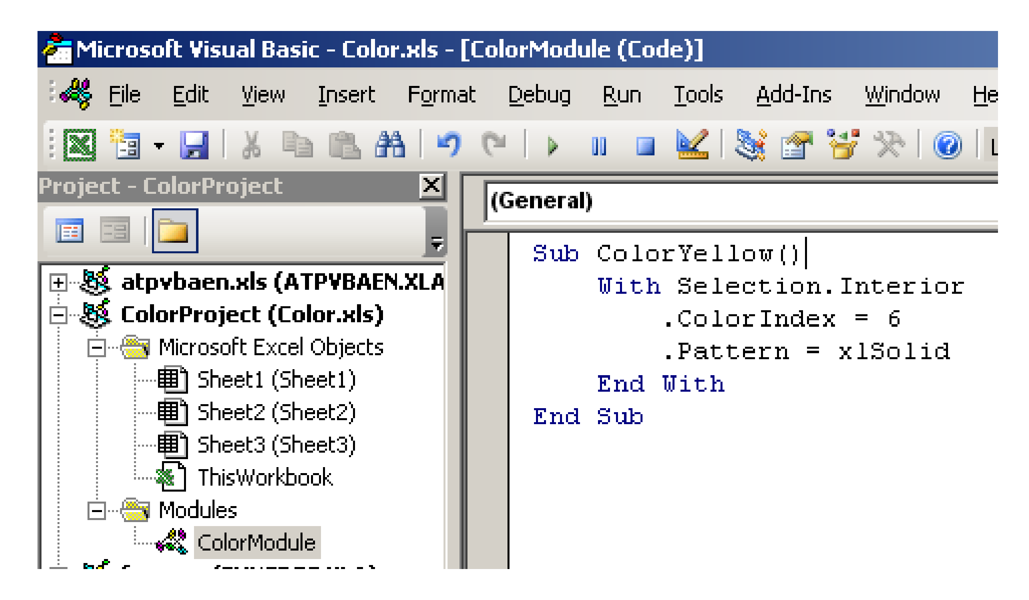
XLSTART du dossier d'installation de MS Office de son ordinateur (copiez les fichiers *.dot dans le dossier STARTUP)

7.6 Macros référencées

Il est possible depuis un nouveau projet V.B.A. de faire référence à une routine V.B.A. se trouvant dans un module d'un fichier extérieur (xls ou xla).

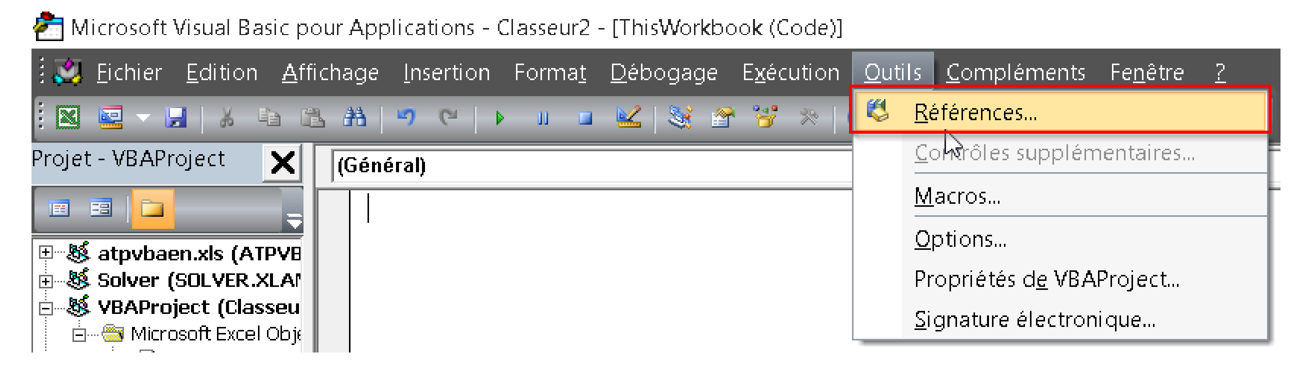
Remarque: Il est quand même conseillé de plutôt faire les manipulations ci-dessous avec des fichiers XLA (qui par défaut ne s'affichent pas à l'utilisateur final).

La méthode d'utilisation est simple. Faisons un exemple en supposant que nous avons enregistré une macro qui change la couleur d'une cellule en jaune:

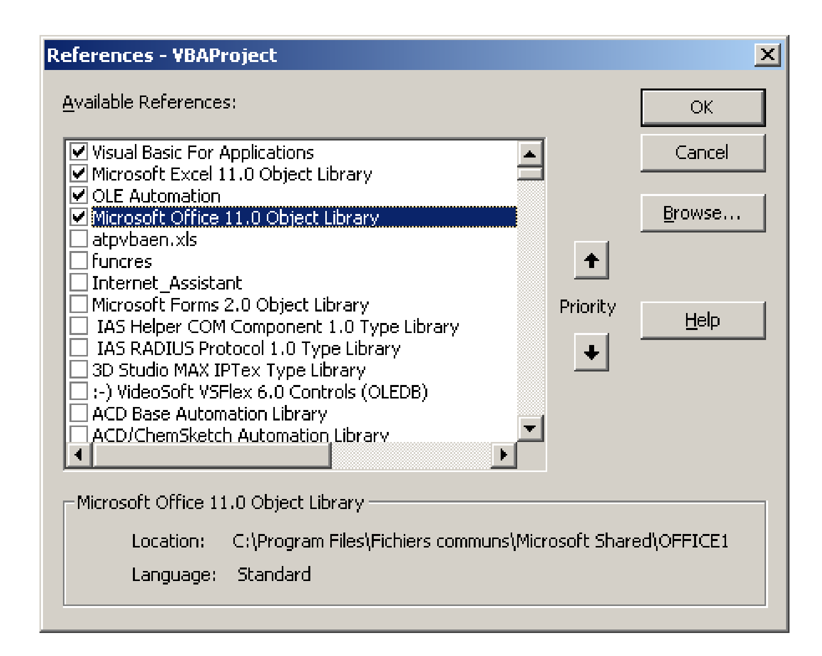


dans un fichier Excel nommé Color.xls (attention à renommer le nom du module et du projet V.B.A. avant de continuer !!!).

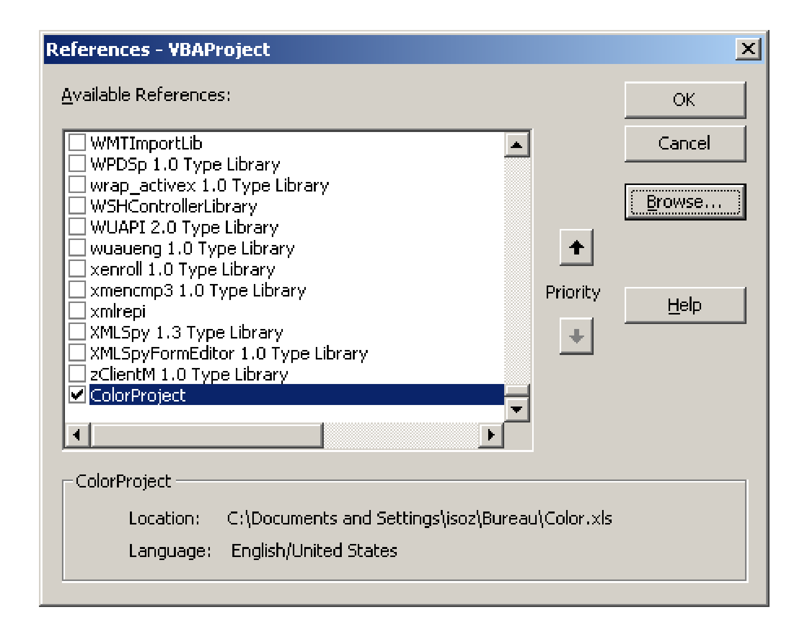
Ensuite, dans le classeur qui doit utiliser cette macro externe, vous ouvrez l'éditeur Visual Basic Application et dans le menu **Outils** vous cliquez sur l'option **Références**:



Il vient alors:



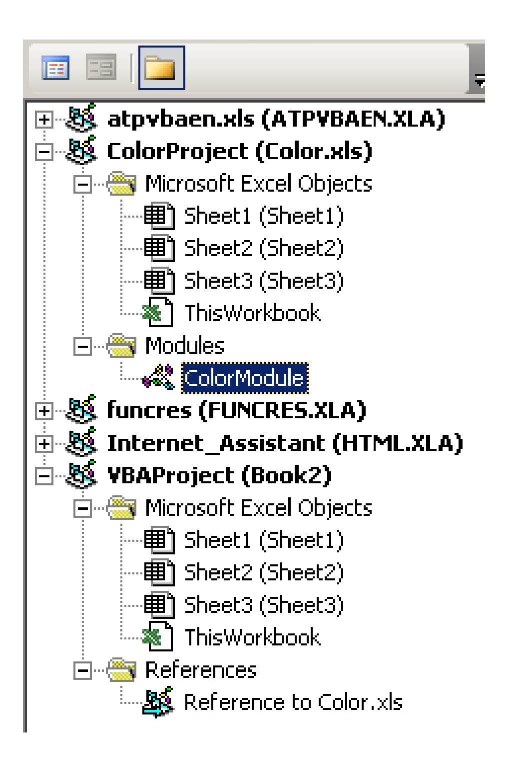
vous cliquez ensuite sur le boutton *Parcourir* pour charger le fichier Excel Color.xls de manière à obtnir le résultat suivant:



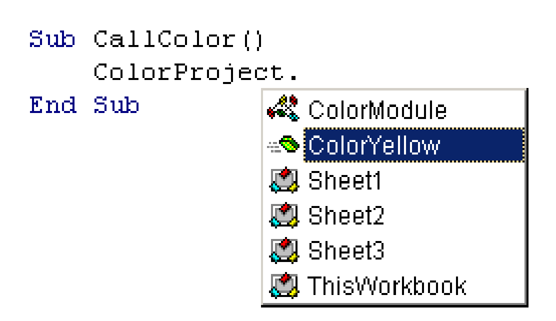
Ensuite dans l'explorateur de projet nous pouvons observer:

1. Que le projet *ColorProject* est dans la liste

2. Qu'une référence externe (*Reference to Color.xls*) a été rajoutée dans le projet du nouveau fichier Excel en cours



Il ne reste plus qu'à utiliser la routine *ColorYellow* depuis le nouveau classeur actif:



On peut en V.B.A. afficher la liste des références activées assez simplement:

```
Sub ListReferences()
   Dim Ref As Reference
   Msg = ""
   For Each Ref In ActiveWorkbook.VBProject.References
      Msg = Msg & Ref.Name & vbNewLine
      Msg = Msg & Ref.Description & vbNewLine
      Msg = Msg & Ref.FullPath & vbNewLine & vbNewLine
   Next Ref
   MsgBox Msg
End Sub
```

Il est aussi possible de d'ajouter automatiquement nos propres références (fonctionnalité extrêmement importante dans la pratique!):

```
Sub AjoutReference()
   Dim Ref As Object
   Set Ref = Application.VBE.ActiveVBProject.References
   On Error Resume Next
   Ref.AddFromFile "StdOle2.tlb"
   Ref.AddFromFile "Fm20.dll"
End Sub
```

7.7 Exercice facultatif

Ne pas oublier d'enregistrer votre fichier toutes les 10 minutes (mais pas pendant l'enregistrement de la macro) !!!!

Faire l'interface sur la feuille + sur une barre d'outils attachée avec tous les boutons et menus nécessaires.

Vous devez:

1. Créer un classeur que vous nommerez *GestionDatas.xls*

2. Importer les données du tableau des enregistrements (mis à disposition par votre formateur) se trouvant selon votre choix dans MS Access (faire par liaison) ou MS Excel (par copier/coller, avec ou sans liaisons).

   Pour ceux ayant choisi la liaison avec MS Access (base JET), faire déjà une macro permettant d'effectuer la mise à jour de la liaison (et donc des données).

3. Faire une macro pour la saisie, la modification et la recherche plus la suppression de données à l'aide de l'outil « grille » de MS Excel (ne refaites pas la roue!)

4. Faire en sorte que l'utilisateur puisse imprimer selon son choix un format donné de la table des enregistrements (ne pas oublier de faire la mise en page: en-têtes et pieds de page, etc…). L'utilisateur doit avoir pouvoir imprimer les colonnes des *Articles*, des *Secteurs* et *Etats factures*. Pour la colonne *Etats factures*, il doit pouvoir à l'aide d'une liste déroulant choisir lesquelles il désire impriment {Oui, Non, -}.

5. Proposer à l'utilisateur trois vues personnalisées pour l'aperçu avant impression. Idée pour ceux qui utilisent la table d'enregistrements MS Excel):

   4.1 Tous les champs

   4.2 Champs N°client+ Quantité +Prix total

   4.3 Tout sauf N° article + Secteur + Facture payée

6. Créer une liste déroulante contenant les *N° clients* qui se met à jour (effectivement il peut y avoir de nouveaux clients…) lorsque l'utilisateur le décide (par un bouton). L'utilisateur doit avoir aussi la possibilité de choisir « tous » dans la liste !!!

   5.1 Créer une feuille avec une « jolie » table de statistique qui en fonction du client choisi affiche dans la liste créée précédemment:

   1\. Le nombre de commandes effectuées par ce client

   2\. La quantité totale commandée (et la somme correspondante)

   3\. Le nombre de factures payées (et la somme correspondante)

4. Le nombre de factures non payées (et la somme correspondante)

5. La commande ayant été la plus coûteuse – globalement pour tous les clients (et le nom de l'article correspondant)

6. La commande ayant été la moins coûteuse – globalement pour tous les clients (et le nom de l'article correspondant)

7. La valeur moyenne de la somme payée pour toutes les commandes – globalement pour tous les clients

8. L'article le plus commandé par ce client.

5.2 Créer une case à cocher qui affiche (respectivement masque) le meilleur client dans une cellule rouge de votre choix de la feuille intro (le meilleur au sens du nombre de commandes et non du cash-flow). Question subsidiaire: une macro est-elle nécessaire pour cela???

7. Permettre à l'utilisateur d'afficher en aperçu avant impression un graphique (histogramme) avec en abscisses les secteurs d'activités et en ordonnées sur 2 axes: (1) les nombre d'unités vendues (2) la somme correspondante et ceci pour un ou tous les clients !!! L'utilisateur doit pouvoir choisir s'il désire visualiser le graphique avec toutes les commandes, seulement les commandes payées ou non payées. Il doit également pouvoir choisir s'il désire voir le nombre d'unités ou pas sur le graphique.

La feuille sur laquelle se trouve le graphique devra se nommer "Graphique".

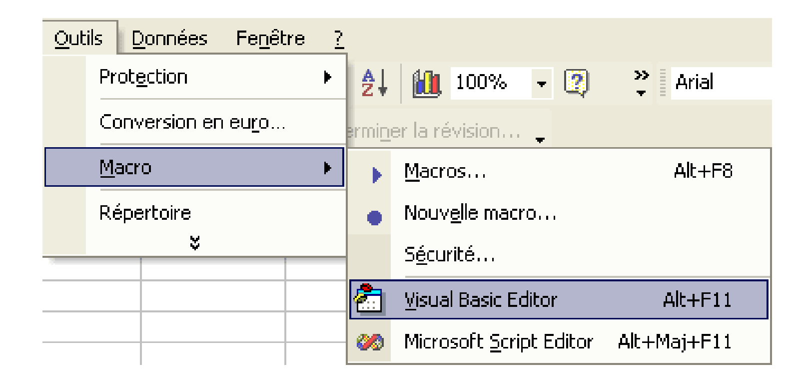
8. Créer et finir l'interface d'utilisation et créer la barre d'outils d'utilisation de votre application.

9. Créer une macro complémentaire …. !!!! … qui dans le classeur où elle est exécutée sélectionne une feuille qui doit s'appeler graphique et exporte cette feuille en page web dynamique à la racine du disque c:/ et l'ouvre ensuite automatiquement dans le navigateur Internet par défaut de la machine de l'utilisateur.

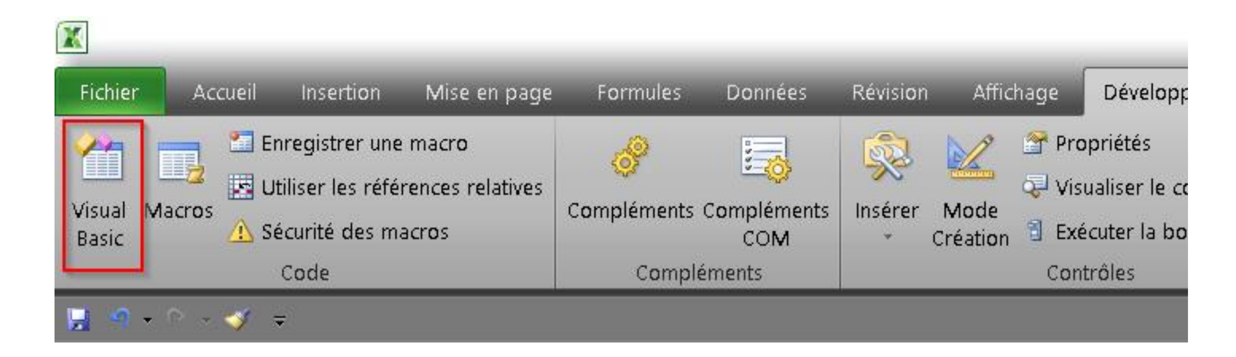
Durée de l'exercice: entre 1 jour et 1 ½ jour

8. ÉDITEUR VISUAL BASIC APPLICATION

L'éditeur de macro, ou V.B.E. (Visual Basic Editor) est l'environnement de programmation de V.B.A. Il se lance par le menu **Outils/Macro/Visual Basic Editor** pour MS Excel 2003 et antérieur:

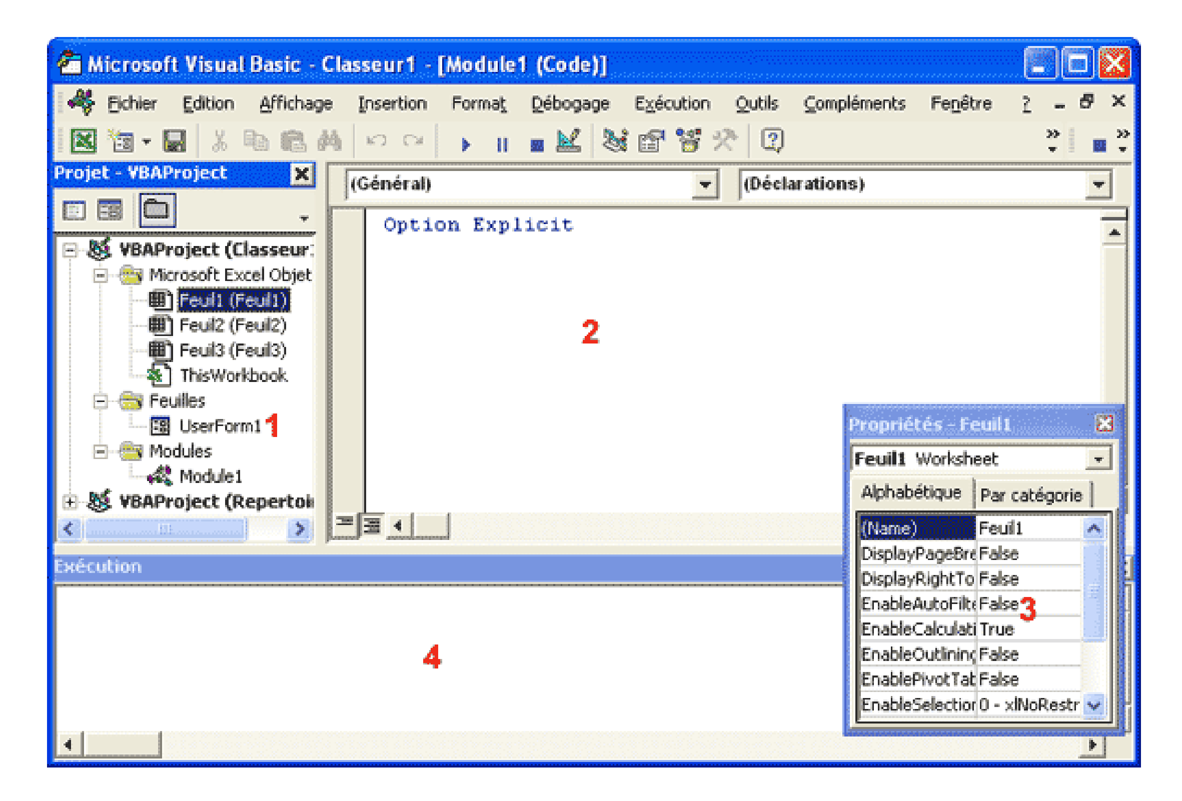


Ou simplement par le même bouton se trouvant dans l'onglet **Développeur** de MS Excel 2007 et ultérieur:



Ou dans le cas général par le raccourci **Alt+F11**.

Voici un mini-descriptif de la fenêtre du VBAE:



1 - Fenêtre VBAProject. Elle présente les différents projets ouverts et permet de naviguer facilement entre vos différentes feuilles de codes V.B.A.

2 - Fenêtre Code. C'est l'endroit où vous allez saisir votre code V.B.A.

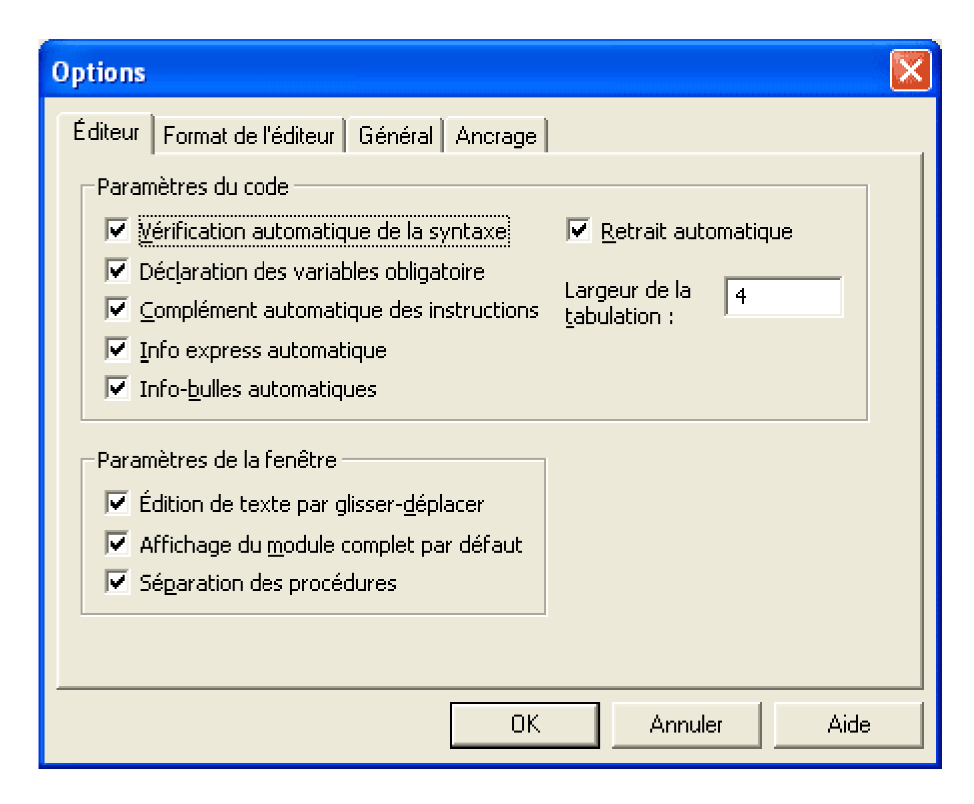
3 - Fenêtre Propriétés. Propriétés de l'objet sélectionné.

4 - Fenêtre Exécution. Elle permet de tester une partie du code. Elle peut s'avérer très utile pour voir comment s'exécutent certaines lignes de code.

Il est fort probable que l'aspect de votre éditeur de macros soit différent. Il est en effet personnalisable car chaque fenêtre peut être masquée puis réaffichée par le menu **Affichage**. Cependant, cette configuration vous permet de débuter de façon confortable l'écriture de vos premières macros.

Il est donc important de bien configurer l'éditeur de macros. En effet, VBAE peut vous aider dans l'écriture de votre code et le mettre en forme de façon à ce qu'il soit plus facile à lire.

Sous VBAE, lancer le menu **Outils/Options**:



Vérification automatique de la syntaxe:

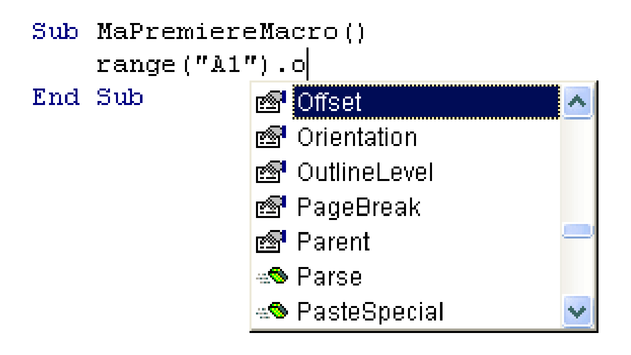
Vérification automatiquement de la syntaxe lors de la saisie d'une ligne de code.

Déclarations de variables obligatoires:

Sous VBA, la déclaration de variables n'est pas obligatoire. Cependant, nous vous conseillons de cocher cette option. De plus amples informations au sujet des variables seront disponibles dans le cours "Les variable". Si la case est cochée, l'instruction "`Option Explicit`" est ajoutée dans les déclarations générales de tout nouveau module.

Complément automatique des instructions:

Cette option permet à V.B.E. de vous aider dans la saisie de votre code:



Vous comprendrez très vite son utilité lorsque vous saisirez vos premières lignes de codes.

Info express automatique:

Encore une option très utile. Elle affiche les différents arguments que possède la fonction que vous venez de taper:

```
Sub MaPremiereMacro()
    range(
End  Range(Cell1, [Cell2]) As Range
```

Info-bulles automatique:

Indispensable lors d'un débogage pas à pas. Elle permet l'affichage de la valeur de vos variables.

Retrait automatique:

Permet à VBAE de placer chaque ligne de code au même niveau que la ligne précédente. Le retrait de lignes se fait par les touches **Tab** et **Shift+Tab**. Cette option est nécessaire pour une bonne lecture du code V.B.A.

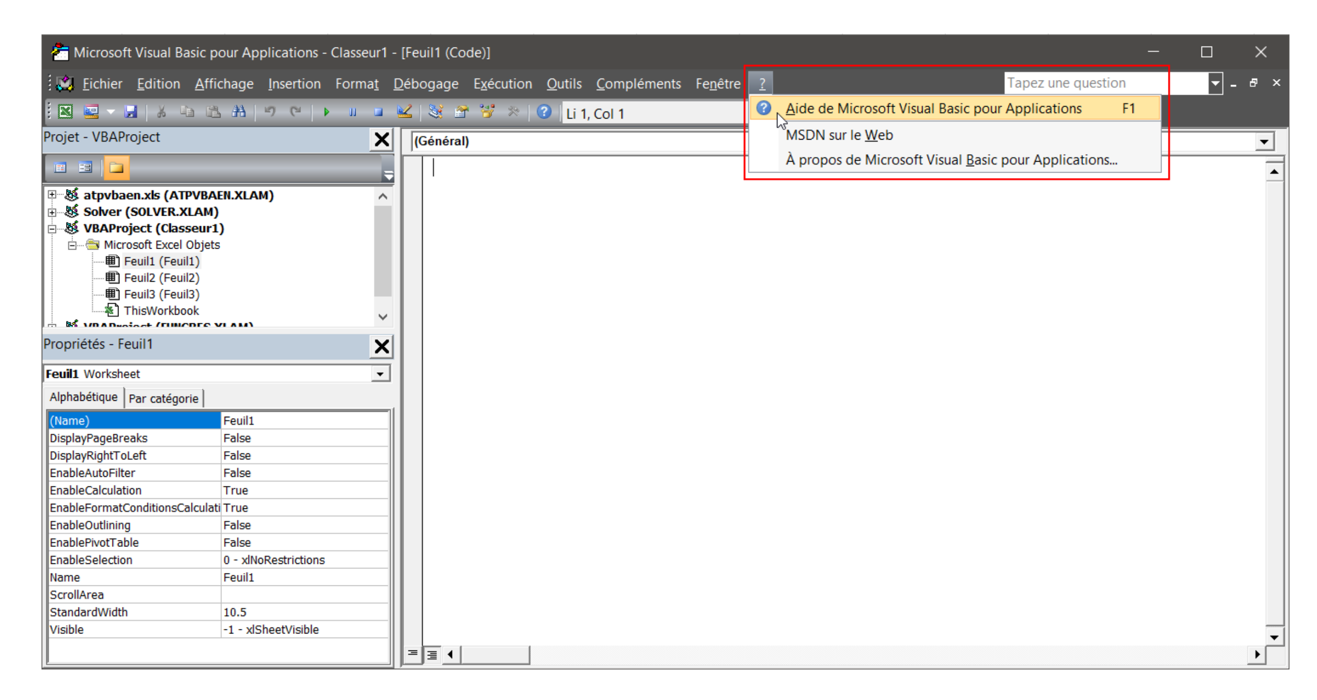
Paramètres de la fenêtre:

Les 3 options sont intéressantes. L'édition de texte par glisser-déplacer permet de déplacer à l'aide de la souris le bloc de code sélectionné, l'affichage du module complet par défaut permet l'affichage de toutes les procédures d'un même module et la séparation des procédures oblige VBAE à créer des traits entre chaque procédures.

Les autres onglets sont évidents à comprendre. Avec l'expérience vous comprendrez par vous-même de quoi il s'agit.

8.1.1 Aide

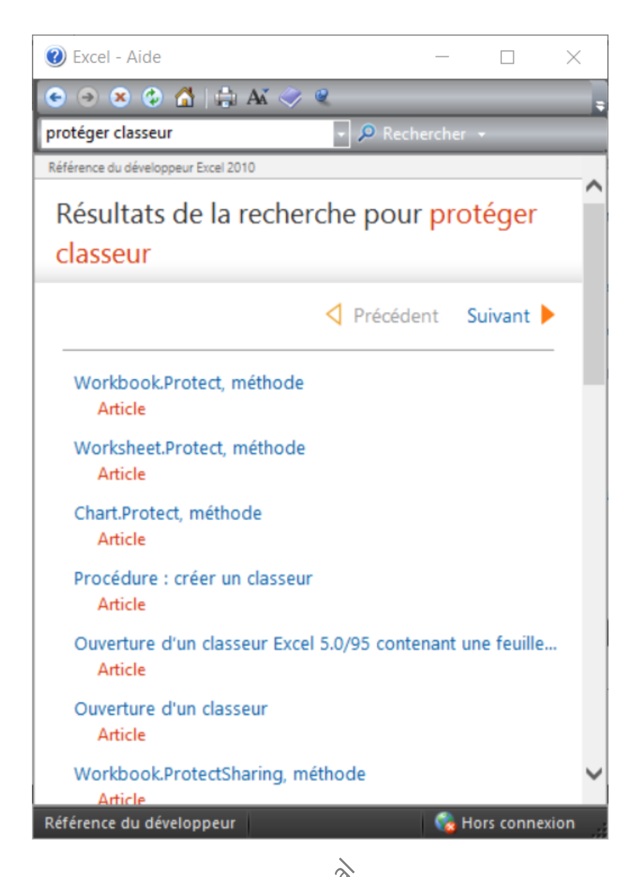
Signalons aussi le non mois important bouton d'aide:



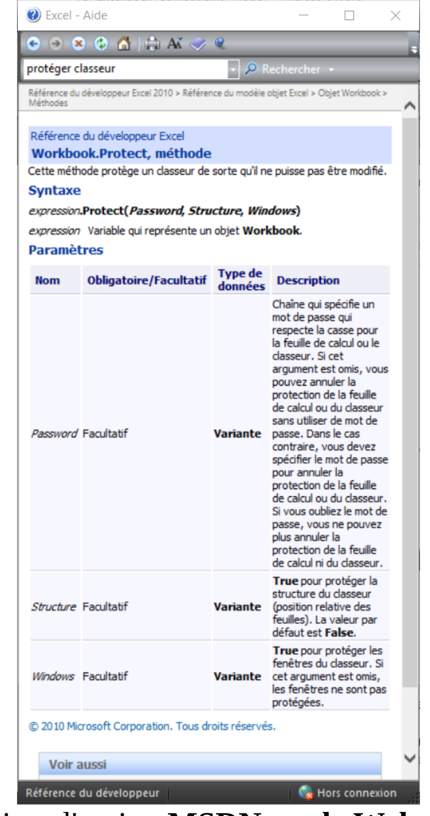
La première option **Aide de Microsoft Visual Basic pour Application** est assez évidente et similaire à n'importe quel autre logiciel:



On peut y chercher de l'aide sur des objectifs à atteindre:



Et consulter l'article correspondant (qui n'est pas toujours d'une grande utilité pour ceux qui n'ont pas plusieurs années de pratique en VBA):



Référence du développeur Excel

Workbook.Protect, méthode

Cette méthode protège un classeur de sorte qu'il ne puisse pas être modifié.

Syntaxe

expression.**Protect(*Password, Structure, Windows*)**

expression Variable qui représente un objet **Workbook**.

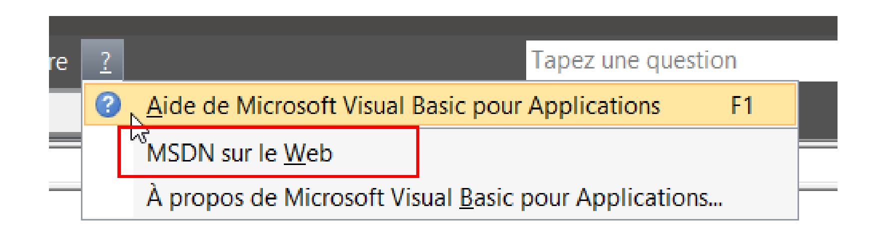
Paramètres

Nom	Obligatoire/Facultatif	Type de données	Description
Password	Facultatif	**Variante**	Chaîne qui spécifie un mot de passe qui respecte la casse pour la feuille de calcul ou le classeur. Si cet argument est omis, vous pouvez annuler la protection de la feuille de calcul ou du classeur sans utiliser de mot de passe. Dans le cas contraire, vous devez spécifier le mot de passe pour annuler la protection de la feuille de calcul ou du classeur. Si vous oubliez le mot de passe, vous ne pouvez plus annuler la protection de la feuille de calcul ni du classeur.
Structure	Facultatif	**Variante**	**True** pour protéger la structure du classeur (position relative des feuilles). La valeur par défaut est **False**.
Windows	Facultatif	**Variante**	**True** pour protéger les fenêtres du classeur. Si cet argument est omis, les fenêtres ne sont pas protégées.

© 2010 Microsoft Corporation. Tous droits réservés.

Voir aussi

Sinon il y a plus important, qui est l'option **MSDN sur le Web**:



Ce qui nous amène à:

https://msdn.microsoft.com/fr-fr/vba/vba-language-reference
Référence du langage VBAL...
Microsoft | Developer Network
Abonnements MSDN
Obtenir des outils
Centre de développement
Office VBA Reference > Language Reference VBA > Référence du langage VBALanguage ref...
Partage
Table des matières
Présentation de la référence du langage VBA
Concepts
Rubriques de procédures
Référence du langage
Glossaire
Cet article a été traduit par une machine. Vous pouvez afficher le texte en anglais dans une fenêtre contextuelle en déplaçant le pointeur de la souris sur le texte.
Référence du langage VBA
office 365 dev account | Dernière mise à jour: 29/09/2017 | 1 Contributeur
Bienvenue dans le Visual Basic pour Applications de référence du langage (VBA) pour Office.Cette référence contient des vues d'ensemble conceptuelles, des tâches de programmation, exemples et références pour vous guider dans le développement de solutions basées sur VBA.
Remarque Vous souhaitez développer des solutions qui étendent l'expérience Office sur plusieurs plates-formes?Découvrez le nouveau modèle de compléments Office.
COMMENTAIRES

Ce qui après une recherche sur le même sujet donnera:

https://msdn.microsoft.com/fr-fr/vba/excel-vba/articles/worksheet-protect-method

Worksheet.Protect, méthod...

Microsoft | Developer Network | Abonnements MSDN | Obtenir des outils

Centre de développement

Office VBA Reference > Excel VBA > Worksheet.Protect, méthode (Excel) | Partage

Table des matières
- Présentation de VBA Excel
- Concepts
- Modèle objet
  - AboveAverage, objet
  - Action, objet
  - Actions, objet
  - AddIn, objet
  - AddIns, objet
  - AddIns2, objet
  - Adjustments, objet
  - AllowEditRange, objet
  - AllowEditRanges, objet
  - Application, objet
  - Areas, objet
  - AutoCorrect, objet

Worksheet.Protect, méthode (Excel)

office 365 dev account | Dernière mise à jour: 04/10/2017 | 1 Contributeur

DANS CET ARTICLE +

Cette méthode protège une feuille de calcul de sorte qu'elle ne puisse être modifiée.

Syntaxe

expression . **Protect**(***Password***, ***DrawingObjects***, ***Contents***, ***Scenarios***, ***UserInterfaceOnly***, ***AllowFormattingCells***, ***AllowFormattingColumns***, ***AllowFormattingRows***, ***AllowInsertingColumns***, ***AllowInsertingRows***, ***AllowInsertingHyperlinks***, ***AllowDeletingColumns***, ***AllowDeletingRows***, ***AllowSorting***, ***AllowFiltering***, ***AllowUsingPivotTables***)

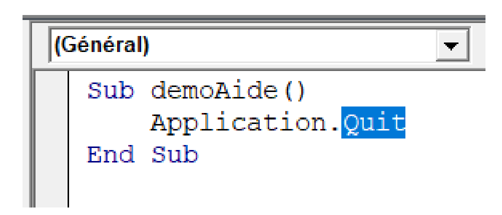
expression Variable qui représente un objet **Worksheet**.

Paramètres

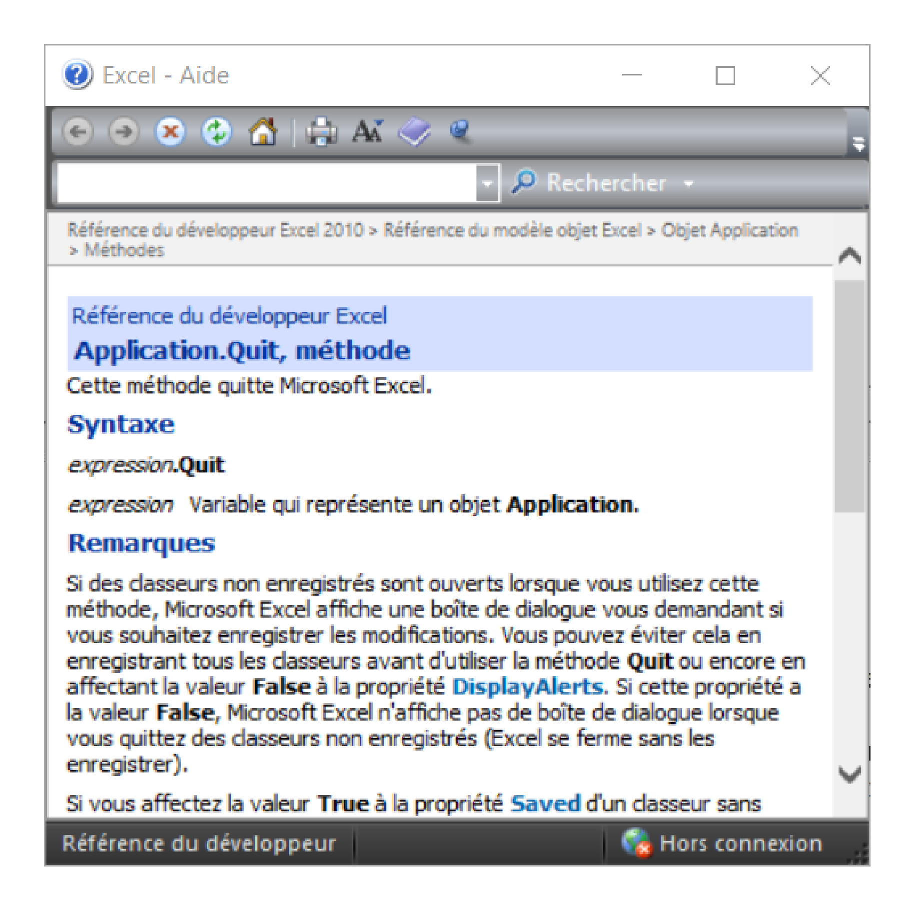
Nom	Requis/Facultatif	Type de données	Description

COMMENTAIRES

Signalons aussi que lorsqu'on écrit un code VBA on peut en sélectionnant une propriété ou méthode ou classe:



et ensuite en appuyant sur la touche F1 du clavier, avoir automatiquement l'aide correspondante qui apparaît:

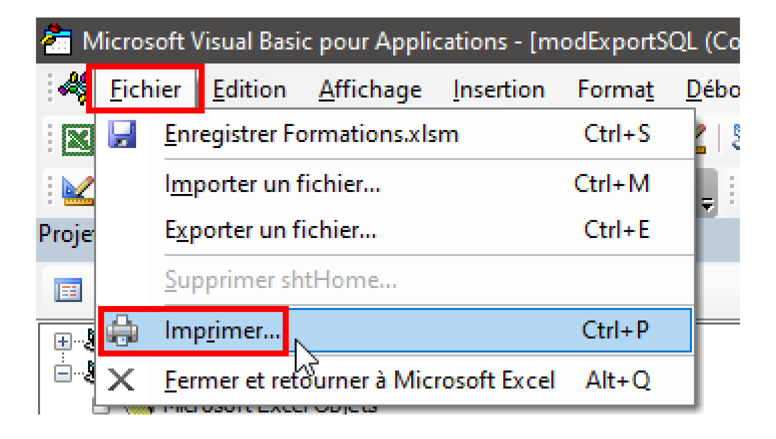


8.1.2 Imprimer

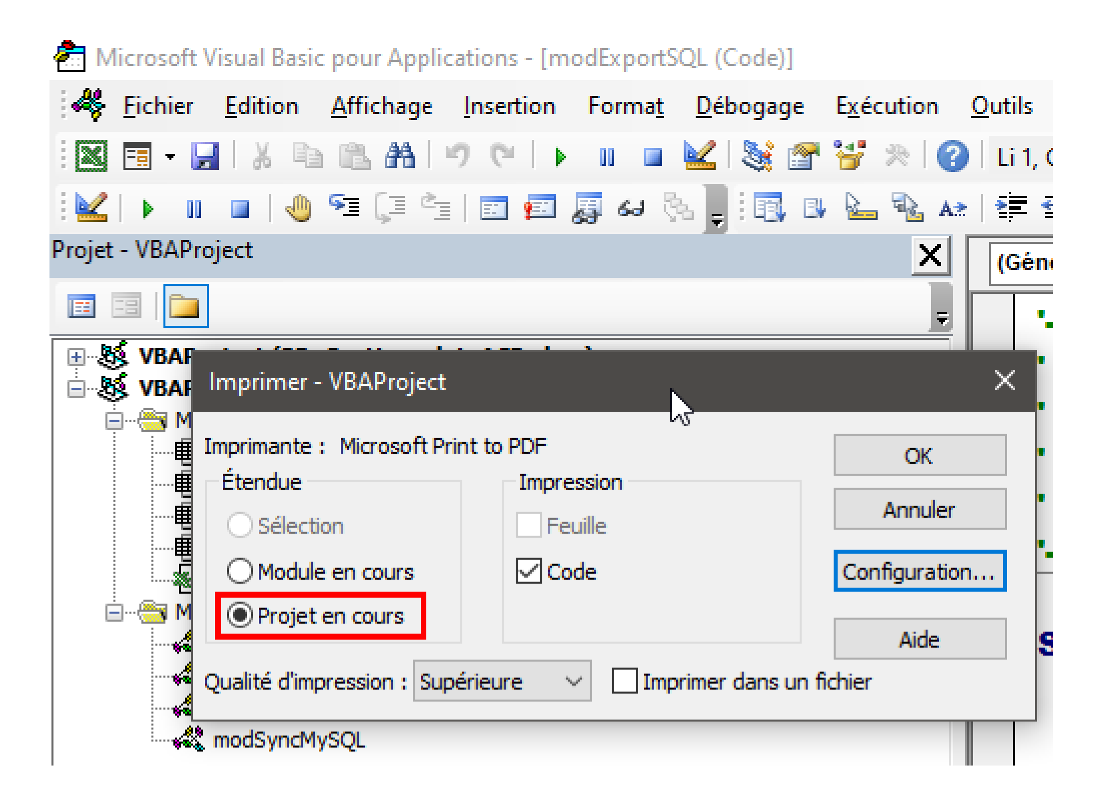
Lorsque vous héritez d'un fichier ou que vous êtes un consultant qu'on demande de travailler sur un fichier Microsoft Office avec du VBA et des macros il est toujours bon d'estimer la quantité de code à analyser de façon globale.

Malheureusement il n'existe pas de manière native un compteur de ligne de code dans la VBAE, mais une solution consiste à imprimer tout le code sur des pages A4 (en PDF), ce qui donne souvent une bonne idée de la quantité de travail qu'il va falloir fournir pour comprendre l'ensemble du code.

Pour imprimer tout un projet VBA, vous allez donc dans Fichier / Imprimer:



Et dans la boîte de dialogue qui apparaît vous faites bien attention à cocher l'option *Projet en cours*:



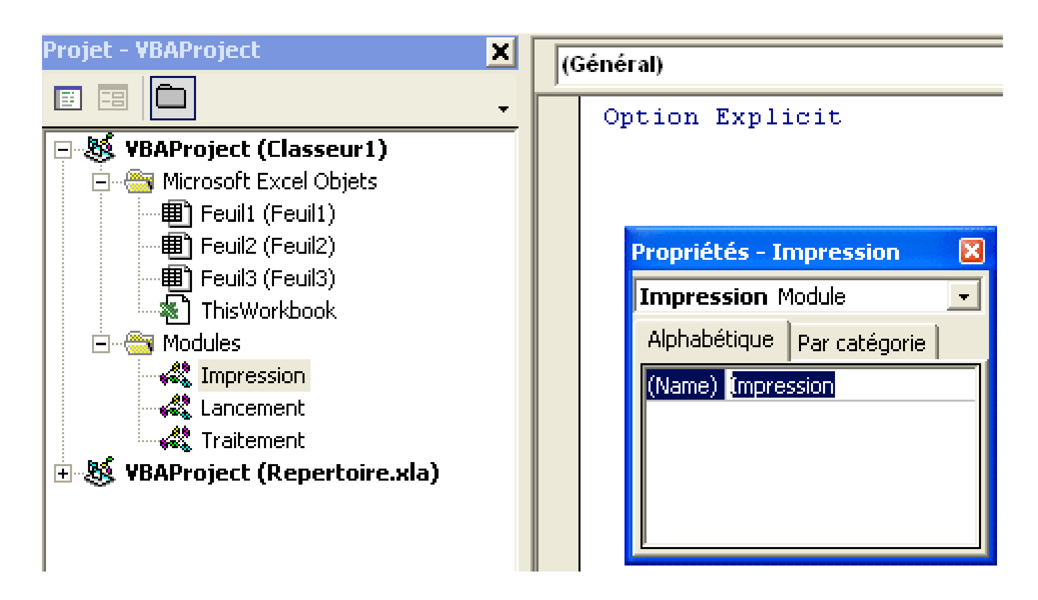
Après quoi l'estimation du temps de travail est quelque chose de très personnel…

Attention!!! Par défaut la taille de police imprimée sera la taille de police choisie pour l'éditeur VBAE…

8.1.3 Syntaxe des fonctions et procédures

Le code V.B.A. s'écrit dans les modules à l'intérieur de procédures ou de fonctions.

Dans VBAE, créez un nouveau module par le menu **Insertion/Module**. Renommez le module à l'aide de la fenêtre propriétés, la recherche de vos procédures sera plus rapide.



Une procédure est une suite d'instructions effectuant des actions. Elle commence par Sub + NomDeLaProcédure et se termine par End Sub. Le nom des procédures ne doit pas commencer par une lettre et ne doit pas contenir d'espaces. Utilisez le caractère de soulignement pour séparer les mots. Nous vous conseillons de les écrire comme des noms propres.

Pour déclarer une procédure, taper Sub et son nom puis taper Entrée. VBAE ajoute automatiquement les parenthèses et la ligne End Sub.

Exemple de Procédure nommée Essai:

```
Sub Essai()
    MsgBox "Hello World!"
    Application.Speech.Speak Text:="Hello World!"
End Sub
```

Une fonction est une procédure qui renvoie une valeur. Elle se déclare de la même façon qu'une procédure.

Exemple de fonction nommée Calcul:

```
Function Calcul(Nbre1 As Integer, Nbre2 As Integer)
    Calcul = Nbre1 + Nbre2
End Function
```

En général, on écrit une instruction par ligne. Il est cependant possible d'écrire plusieurs instructions sur une même ligne en les séparant par le caractère «: ».


```
Sub Essai()
    Nbre1 = 1: Nbre2 = 2
End Sub
```

On peut également appeler une fonction à partir d'une procédure:

```
Sub AppelFonction()
    MsgBox Calcul(5,3)
End Sub
```

ou encore si nous avions:

```
Sub Calcul(byVal Nbre1 As Integer,byVal Nbre2 As Integer)
    Resultat = Nbre1 + Nbre2
    MsgBox Resultat
End sub

Sub AppelRoutine()
    Nbre1=1
    Nbre2=2
    Calcul Nbre1, Nbre2
End sub
```

Nous voyons que la technique (syntaxe) entre une procédure qui appelle une fonction et une procédure qui appelle une fonction sont donc nettement différentes. Il convient d'y prendre garde.

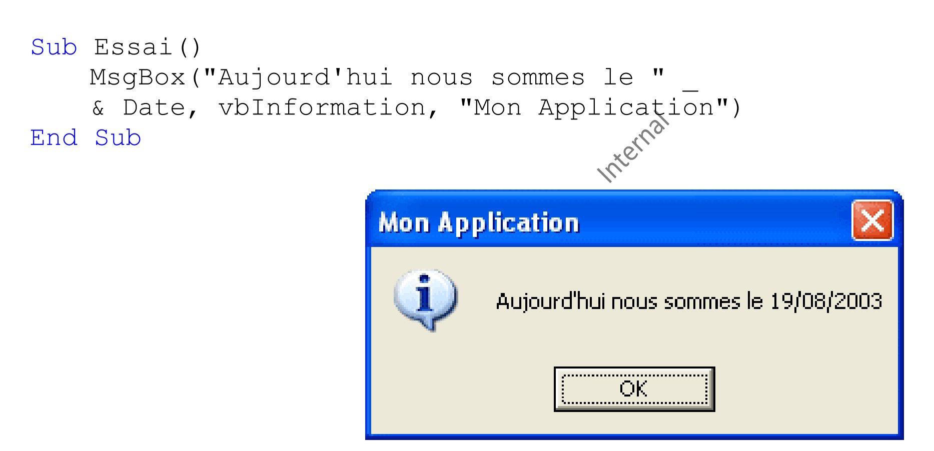
Il est possible d'ajouter des lignes de commentaire entre les lignes d'instruction ou au bout de celles-ci. Les commentaires sont précédés d'une apostrophe et prennent une couleur différente (définie dans les options de VBAE):

```
Sub Essai()
    Dim Invite as String 'Nom de l'utilisateur
    Invite = "Toto"
    'Message bonjour à l'utilisateur
    MsgBox "Bonjour " & Invite
End Sub
```

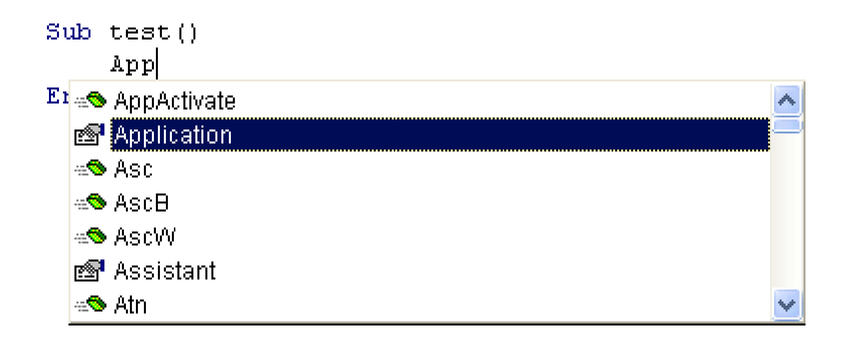


Il n'y a pas de limite de caractères pour chaque ligne d'instruction. Il est toutefois possible d'écrire une instruction sur plusieurs lignes afin d'augmenter la visibilité du code. Pour cela, il faut ajouter le caractère de soulignement avant le passage à la ligne (touche **Entrée**):

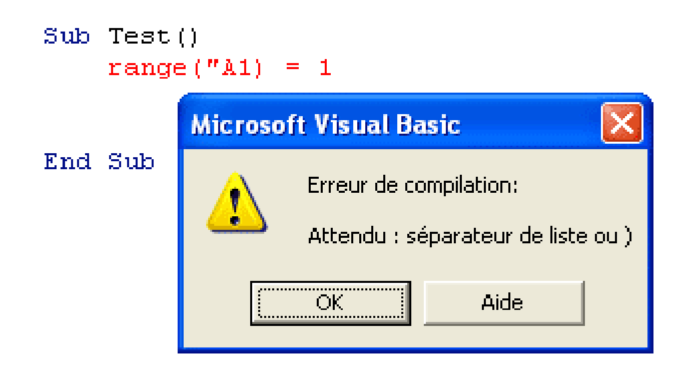
```
Sub Essai()
    MsgBox("Aujourd'hui nous sommes le " _
    & Date, vbInformation, "Mon Application")
End Sub
```

L'option "Info express automatique" permet d'afficher les informations de la fonction que vous venez de taper. Il est également possible d'obtenir de l'aide à tout moment par la combinaison de touches **Ctrl+J**:



La vérification automatique de la syntaxe vous alerte si il y a une erreur dans l'écriture du code et la ligne de code change de couleur. Si la vérification automatique de la syntaxe n'est pas activée, la boite d'alerte ne s'affiche pas.



Chaque procédure Sub ou Function peut être appelée de n'importe qu'elle autre procédure du projet. Pour restreindre la portée d'une procédure au module, déclarez-la en private:

```
Private Sub Essai()
    MsgBox "Bonjour"
End Sub

Private Function Calcul(Nbre1, Nbre2)
    Calcul = Nbre1 + Nbre2
End Function
```

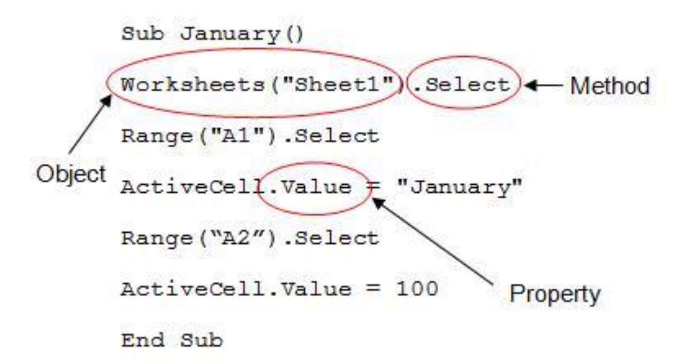
A l'intérieur de vos procédures, écrivez vos instructions en minuscules, V.B.E. se chargera de transformer votre code par des majuscules.

Il existe souvent de multiples façons d'arriver à un résultat. Une bonne analyse des tâches à accomplir est nécessaire avant de se lancer dans la création d'une application.
Si vous n'avez aucune expérience en VBA, vous verrez que l'on y prend vite goût et que l'on arrive très rapidement à de surprenants résultats.
V.B.A. manipule les objets de l'application hôte. Chaque objet possède des propriétés et des méthodes.

8.1.4 Les objets

Chaque objet représente un élément de l'application. Sous MS Excel, un classeur, une feuille de calcul, une cellule, un bouton, etc... sont des objets. Par exemple, Excel représente l'objet Application, Workbook l'objet classeur, Worksheet l'objet feuille de calcul etc...

L'idée est technique de classer le vocabulaire de façon structure en une grammaire ayant une certaine cohérence:



Tous les objets de même type forment une collection comme, par exemple, toutes les feuilles de calcul d'un classeur. Chaque élément est alors identifié par son nom ou par un index.

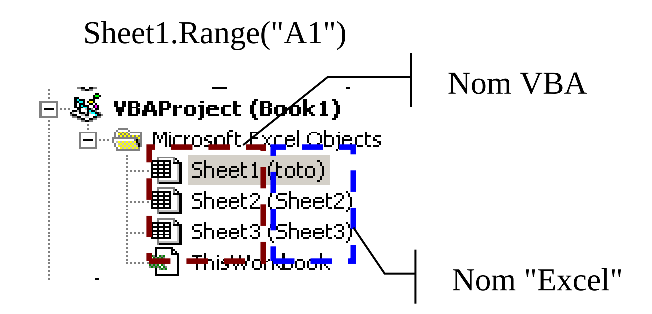
Pour faire référence à la Feuil2, on va utiliser Worksheets(2) ou Worksheets("Feuil2")

Chaque objet peut avoir ses propres objets. Par exemple, Excel possède des classeurs qui possèdent des feuilles qui possèdent des cellules. Pour faire référence à une cellule, on pourrait ainsi utiliser:

Application.Workbooks(1).Worksheets("Feuil2").Range("A1")

ou mieux encore (en utilisant le nom VBA):

Sheet1.Range("A1")



8.1.5 Les propriétés

Une propriété correspond à une particularité de l'objet. La valeur d'une cellule, sa couleur, sa taille, etc...sont des propriétés de l'objet Range. Les objets sont séparés de leurs propriétés par un point. On écrira ainsi Cellule.Propriété=valeur:

```
'Mettre la valeur 10 dans la cellule A1
Range("A1").Value = 10
```

Une propriété peut également faire référence à un état de l'objet. Par exemple, si on veut masquer la feuille de calcul "Feuil2", on écrira:

```
Worksheets("Feuil2").Visible = False
```

ce qui peut aussi s'écrire:

```
Feuil2.Visible = xlSheetHidden
```

mais pour les deux méthodes, la feuille sera toujours visible dans le menu **Format/Feuille/Afficher**. Pour bloquer cela il faudra utiliser une autre commande:

```
Feuil2.Visible = xlSheetVeryHidden
```

ou pour renommer le nom VBA (codename) de la feuille active:

```
ThisWorkbook.VBProject.VBComponents(ActiveSheet.Codename).Name =
"shtFeuille"
```


et il faut pour exécuter cette commande, autoriser l'exécution du VBE dans les paramètres de sécurité de Microsoft Excel.

8.1.6 Les méthodes

On peut considérer qu'une méthode est une opération que réalise un objet. Les méthodes peuvent être considérées comme des verbes tels que ouvrir, fermer, sélectionner, enregistrer, imprimer, effacer, etc... Les objets sont séparés de leurs méthodes par un point. Par exemple, pour sélectionner la feuille de calcul nommé "Feuil2", on écrira:

```
Worksheets("Feuil2").Select
```

Lorsque l'on fait appel à plusieurs propriétés ou méthodes d'un même objet, on fera appel au bloc d'instruction **With** Objet *Instructions* **End With.** Cette instruction rend le code souvent plus facile à lire et plus rapide à exécuter.

```
'Mettre la valeur 10 dans la cellule A1, la police en gras et en italique
et copier la cellule.
With Worksheets("Feuil2").Range("A1")
    .Value = 10
    .Font.Bold = True
    .Font.Italic = True
    .Font.Color = vbRed
    .Font.Name = "Arial"
    .Copy
End With
```

Ce vocabulaire peut paraître déroutant mais deviendra très rapidement familier lors de la création de vos premières applications.

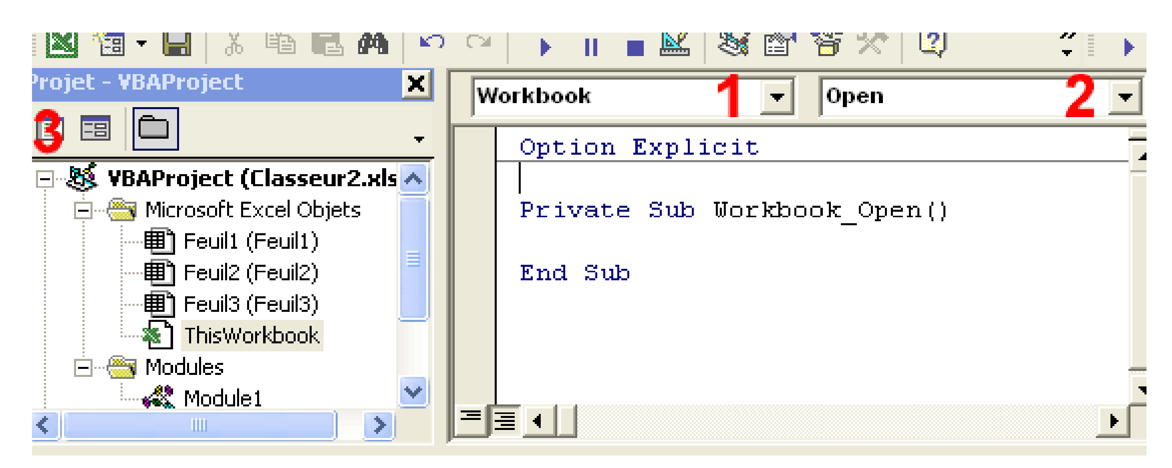
8.1.7 Les événements

Pour qu'une macro se déclenche, il faut qu'un évènement (un clic sur un bouton, l'ouverture d'un classeur, etc...) se produise. Sans évènements, rien ne peut se produire.

Les principaux objets pouvant déclencher une macro sont:

1. Un classeur
2. Une feuille de travail
3. Une boite de dialogue

Chacun de ces objets possède leur propre module. Pour y accéder, lancer l'éditeur de macro:



Pour créer une procédure évènementielle liée à un classeur, sélectionner le classeur "ThisWorkbook" puis cliquez sur l'icône **3** (ou plus simplement double-clic sur "ThisWorkbook").

Vous accédez ainsi au module lié à l'objet. Sélectionnez "Workbook" dans la liste **1** puis sur l'évènement désiré dans la liste **2**.

Par exemple, le code suivant lancera la procédure nommée "Test" à l'ouverture du classeur:

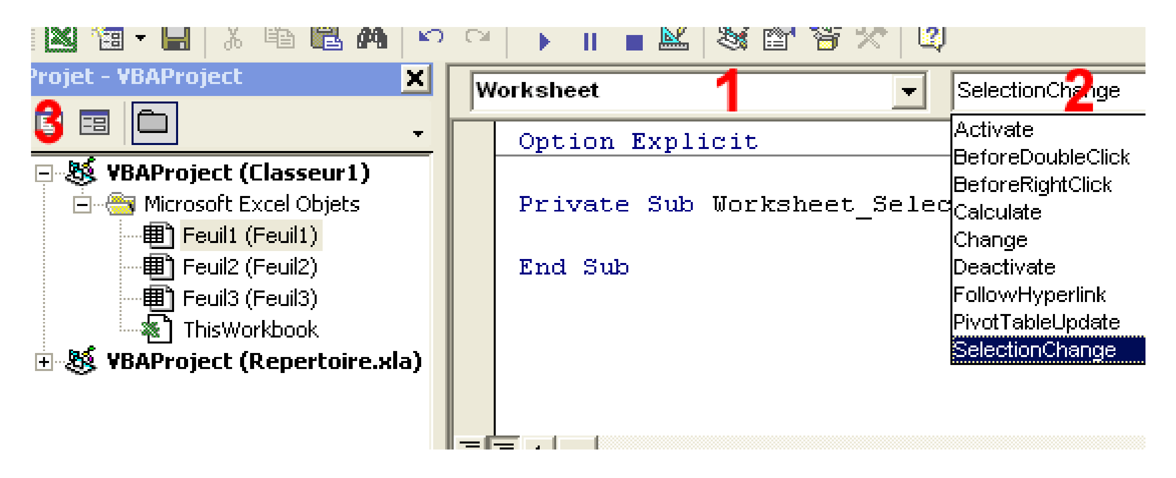
```
Private Sub Workbook_Open()
    Test
End Sub
```

Liste des évènements de l'objet Workbook:

Evénements	Se produit
Activate	quand le classeur ou une feuille est activé
AddinInstall	quand le classeur est installé en macro complémentaire
AddinUninstall	quand le classeur est désinstallé en macro complémentaire

BeforeClose	avant que le classeur soit fermé
BeforePrint	avant l'impression du classeur
BeforeSave	avant l'enregistrement du classeur
Deactivate	quand le classeur ou une feuille est désactivé
NewSheet	lorsqu'une nouvelle feuille est créée
Open	à l'ouverture du classeur
PivotTableCloseConnection	lorsqu'un qu'un rapport de tableau croisé dynamique se déconnecte de sa source de données
PivotTableOpenConnection	lorsqu'un qu'un rapport de tableau croisé dynamique se connecte à une source de données
SheetActivate	lorsqu'une feuille est activée
SheetBeforeDoubleClick	lors d'un double-clic
SheetBeforeRightClick	lors d'un clic avec le bouton droit de la souris
SheetCalculate	après le recalcul d'une feuille de calcul
SheetChange	lors de la modification d'une cellule
SheetDeactivate	lorsqu'une feuille est désactivée
SheetFollowHyperlink	lors d'un clic sur un lien hypertexte
SheetPivotTableUpdate	lors de la mise à jour de la feuille du rapport de tableau croisé dynamique
SheetSelectionChange	lors d'un changement de sélection sur une feuille de calcul
WindowActivate	lorsqu'un classeur est activé
WindowDeactivate	lorsqu'un classeur est désactivé
WindowResize	lors du redimentionnement de la fenêtre d'un classeur

La création d'une procédure évènementielle liée à une feuille de calcul se fait de la même façon.



Liste des évènements de l'objet Worksheet:

Evénements	Se produit

Activate	quand une feuille est activée
BeforeDoubleClick	lors d'un double-clic
BeforeRightClick	lors d'un clic avec le bouton droit de la souris
Calculate	après le recalcul de la feuille de calcul
Change	lors de la modification d'une cellule
Deactivate	quand une feuille est désactivée
FollowHyperlink	lors d'un clic sur un lien hypertexte
PivotTableUpdate	lorsqu'un rapport de tableau croisé dynamique a été mis à jour
SelectionChange	lors d'un changement de sélection

Certaines procédures évènementielles possèdent des paramètres tels que "**Cancel**", qui peut annuler la procédure, "**SaveAsUi**" qui, dans la procédure "Workbook_BeforeSave" affiche la boite "Enregistrer sous", "**Sh**" qui représente la feuille de calcul, "**Target**" qui représente l'objet sélectionné (Cellule, graphique, lien hypertexte), "**Wn**" qui représente la fenêtre active.

Par exemple, le paramètre "Cancel", peut annuler la procédure. Pour empêcher l'impression du classeur, on utilisera:

```
Private Sub Workbook_BeforePrint(Cancel As Boolean)
    Cancel = True
End Sub
```

Pour récupérer la valeur d'une cellule modifiée, on utilisera:

```
Private Sub Worksheet_Change(ByVal Target As Range)
    MsgBox Target.Value
End Sub
```

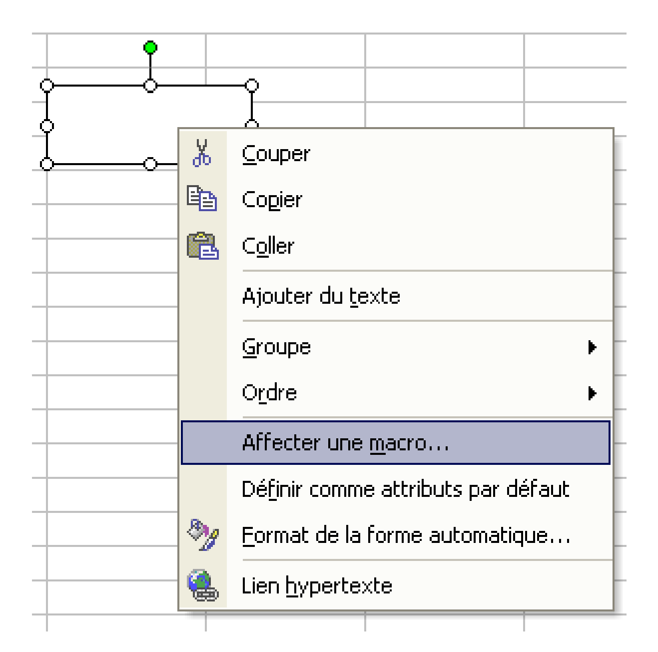
Un cas d'école est hyper classique avec Excel est d'annuler le changement fait par une personne sur une cellule (libre à vous de perfectionner le code ci-dessous) en y remettant la valeur d'origine. Comme l'événement Worksheet_Change n'a pas de paramètre *Cancel* (...) il faut tricher un peu... Voici comment procédér:

```
Dim varOriginalValue As Variant

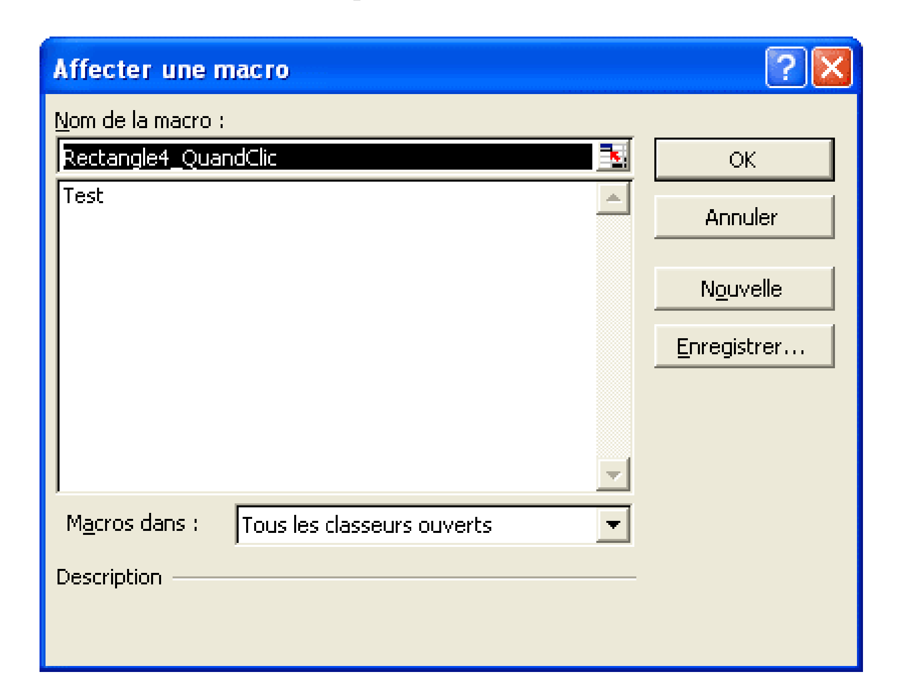
Private Sub Worksheet_Change(ByVal Target As Range)
    'On arrete la gestion des "Change" sinon on tourne en rond
    'car un changement engendre un autre changement et ainsi de suite
    Application.EnableEvents = False
    Target.Value = varOriginalValue
    Application.EnableEvents = True
End Sub

Private Sub Worksheet_SelectionChange(ByVal Target As Range)
    varOriginalValue = Target.Value
End Sub
```

Une macro peut également se déclencher en cliquant sur un élément graphique de l'application (une image, une zone de texte, un objet WordArt, un rectangle ...). Créez un élément puis cliquez sur "Affecter une macro" dans le menu contextuel.



Cliquez sur le nom de la macro désirée puis validez.



Un simple clic sur l'objet lancera la macro.

Il existe également des procédures évènementielles liées aux boites de dialogues (Voir le cours sur les UserForms).

Les évènements non liés aux objets.

Une macro peut également être déclenchée à une heure donnée (OnTime) ou lorsque l'utilisateur appuie sur une touche (OnKey).

Le déclenchement d'une macro nommée "Test" à 15 Heures se fait par la ligne d'instruction suivante:

```
Application.OnTime TimeValue("15:00:00"), "Test"
```

ou si l'on veut que la macro "Test" s'éxecute dans 20 minutes par rapport à maintenant:

```
Application.OnTime Now + TimeValue("00:20:00"), "Test"
```

ou à une date spécifique (à condition que le fichier ne soit pas fermé entre temps):

```
Application.OnTime DateSerial(2012,10,2) + TimeValue("00:00:01"), "Test"
```

Le déclenchement d'une macro nommée "Test" lorsque l'utilisateur appuie sur la touche **F1** se fait par la ligne d'instruction suivante:

```
Application.OnKey "{F1}", "Test"
```

Liste des codes correspondant aux touches:

Touches	Codes
AIDE	{HELP}
ATTN	{BREAK}
BAS	{DOWN}
DÉBUT	{HOME}
DÉFILEMENT	{SCROLLLOCK}
DROITE	{RIGHT}
ÉCHAP	{ESCAPE} ou {ESC}
EFFACER	{CLEAR}
ENTRÉE(pavé numérique)	{ENTER}
ENTRÉE	~
F1 à F15	{F1} à {F15}
FIN	{END}
GAUCHE	{LEFT}
HAUT	{UP}
INSERTION	{INSERT}
PAGE PRÉCÉDENTE	{PGUP}
PAGE SUIVANTE	{PGDN}
RET.ARR	{BACKSPACE} ou {BS}
RETOUR	{RETURN}
SUPPRESSION ou SUPPR	{DELETE} ou {DEL}
TABULATION	{TAB}
VERR.MAJ	{CAPSLOCK}
VERR.NUM	{NUMLOCK}

Il est possible de combiner les touches avec **Alt** en insérant le caractère "%", avec **Ctrl** en insérant le caractère "^" ou avec la touche **MAJ** en insérant le caractère "+". Ainsi le déclenchement d'une macro nommée "Test" lorsque l'utilisateur appuie sur la combinaison de touches **Ctrl+MAJ+F1** se fait par la ligne d'instruction suivante

```
Application.OnKey "^+{F1}", "Test"
```

8.1.8 Types de données

Voici les types de données communes aux logiciels de la suite MS Office:

Le tableau suivant présente les types de données reconnus en précisant la taille des enregistrements et la plage des valeurs.

Type de données	**Taille d'enregistrement**	**Plage**
Byte	1 octet	0 à 255
Boolean	2 octets	True ou False
Integer	2 octets	-32 768 à 32 767
Long (entier long)	4 octets	-2 147 483 648 à 2 147 483 647
Single (à virgule flottante en simple précision)	4 octets	-3.402823E38 à -1.401298E-45 pour les valeurs négatives ; 1.401298E-45 à 3.402823E38 pour les valeurs positives
Double (à virgule flottante en double précision)	8 octets	-1.79769313486231E308 à -4.94065645841247E-324 pour les valeurs négatives ; 4.94065645841247E-324 à 1.79769313486232E308 pour les valeurs positives
Currency (entier à décalage)	8 octets	-922 337 203 685 477.5808 à 922 337 203 685 477.5807
Decimal	14 octets	+/- 79 228 162 514 264 337 593 543 950 335 sans séparateur décimal ; +/-7.9228162514264337593543950335 avec 28 chiffres à droite du séparateur décimal ; le plus petit nombre différent de zéro est +/- 0.0000000000000000000000000001.
Date	8 octets	1er janvier 100 au 31 décembre 9999
Object	4 octets	Toute référence à des données de type Object
String (longueur variable)	10 octets + longueur de la chaîne	0 à environ 2 milliards
String (longueur fixe)	Longueur de la chaîne	1 à environ 65 400
Variant (nombres)	16 octets	Toute valeur numérique, avec la même plage de valeurs qu'une donnée de type Double

Type de données	Taille d'enregistrement	Plage
Variant (caractères)	22 octets + longueur de la chaîne	Même plage de valeurs qu'une donnée de type String de longueur variable
Type défini par l'utilisateur (avec Type)	En fonction des éléments	La plage de valeurs de chaque élément correspond à celle de son type de données.

Le type de données Type peut contenir un ou plusieurs éléments d'un type de données cité ci-dessus, un tableau ou un type de données déjà défini par vous. Pour définir ce genre de type de données, vous devez utiliser l'instruction *Type*, elle doit être définie dans un module, elle peut être privée ou public. Sa syntaxe est la suivante:

```
[Private | Public] Type NomVariable
  [NomElement1 As type]
  [NomElement2 As type]
...
End Type
```

Les mots entre crochet sont facultatifs. Pour mieux comprendre, voici un exemple (l'étoile permet de limiter le nombre de caractères):

```
Type Statistique
   NomStat As String * 50
   Esperance As Single
   Variance As Single
   Mediane As Single
   Kurtosis As Single
   Skewness As Single
End Type
```


Pour utiliser ce genre de type de données, vous déclarez, par exemple, dans une procédure la variable Adherent:

```
Dim Distribution as Statistique
```

La variable Distribution représente donc le nom, le prénom, l'adresse d'une personne. Ensuite, pour attribuer une valeur au nom d'un Adhérent, vous écrivez:

```
Distribution.Nom="Gauss"
```

et ainsi de suite...

Remarque: Quel que soit le type de données, les tableaux nécessitent 20 octets de mémoire, auxquels viennent s'ajouter quatre octets pour chaque dimension et le nombre d'octets occupés par les données. L'espace occupé en mémoire par les données peut être calculé en multipliant le nombre d'éléments par la taille de chacun d'eux. Par exemple, les données stockées dans un tableau unidimensionnel constitué de quatre éléments de type Integer de deux octets chacun occupent huit octets. Ajoutés aux 24 octets d'espace mémoire de base, ces huit octets de données portent la mémoire totale nécessaire pour le tableau à 32 octets. Une variable de type Variant contenant un tableau nécessite 12 octets de plus qu'un tableau seul.

8.1.8.1 Variables de type tableau (arrays)

Le code suivant est un exemple type de l'utilisation de variables de type "Tableau" (ou "array"), il compte le nombre de lignes et colonnes de la sélection pour dimensionner la variable matrice (la cellule active doit être A1):

```
Sub stock_tableau()
    Dim tableau( )
    'Si la dimension est connue on écrira tableau(X) ou tableau(X, Y) dans un cas multidimensionel et tableau(X) as Integer ou tableau(X) as String, etc. si le contenu des composantes est connu.

    nb_lignes = Range("A1").End(xlDown).Row
    nb_colonnes = Range("A1").End(xlToRight).Column
    'on redimensionne (ajout ou suppression!!!) le tableau avec ReDim. Si on veut redimensionner tout en gardant d'éventuelles anciennes valeurs, il faut écrire ReDim Preserve.
    ReDim tableau(nb_lignes, nb_colonnes)
    'pour remplir par déclaration un tel tableau bidimensionnel il suffira d'écrire:
    'tableau([1,1],[2,2],[3,3]) etc…

    For lignes = 1 To nb_lignes
        For colonnes = 1 To nb_colonnes
            tableau(lignes, colonnes) = Cells(lignes, colonnes)
            MsgBox tableau(lignes, colonnes)
        Next colonnes
   Next lignes
   'On peut aussi remplir notre tableau ainsi mais c'est beaucoup moins élégant (suffit de voir la structure du tableau dans le moniteur de variables
   'Tableau = array(Cells(1,1), Cells(nb_lignes,nb_colonnes))
   'On peut mettre le tableau "bien rempli" d'un coup dans une plage de cellules
   Range(cells(1,1),cells(lignes,colonnes)).value=tableau
   'On réinitialise l'array (ici juste pour apprendre la commande)
   Erase tableau()
End Sub
```


Un autre petit exemple simple:

```
Sub ViewTable()
    Dim TabVal As Variant
    Dim i As Integer

    TabVal = Array("Bonjour", 1.244, "=A2+12", "=A3+12")
    For i = 0 To ubound(TabVal)
        Cells(i + 1, 1) = TabVal(i)
    Next i
   'On réinitialise l'array (ici juste pour apprendre la commande)
   Erase tableau()
End Sub
```

ou un autre exemple très utilisé dans la pratique:

```
Sub EcritureTableau()
   Logiciels = Array("Excel", "Word", "PowerPoint", "Outlook", "Access", "Publisher", "OneNote")
   Range("A1:G1") = Logiciels
```

```
   'ou
   Range("A1:A7") = Application.Transpose(Logiciels)
   'On réinitialise l'array (ici juste pour apprendre la commande)
   Erase tableau()

End Sub
```

Encore un exemple très important dans les entreprises:

```
Sub ExtractionMots()
    Dim Tableau() As String
    Dim i As Integer

    'découpe la chaine en fonction des espaces " "
    'le résultat de la fonction Split est stocké dans un tableau
    Tableau = Split("GEN-001;GEN-002;GEN-003", ";")
    'boucle sur le tableau pour visualiser le résultat
    For i = 0 To UBound(Tableau)
      'Le résultat s'affiche dans la fenêtre d'execution de l'éditeur de
macros
        Debug.Print Tableau(i)
    Next i
   'On réinitialise l'array (ici juste pour apprendre la commande)
   Erase tableau()
End Sub
```

Note: Il existe des commandes de conversions de données dans VBA

Voici aussi un code très utile pour trier un tableau en VBA sachant qu'il n'existe pas à ce jour de méthode native simple pour le faire dans Excel:


```
Sub TriTableau()
    Dim arr As Object
    Dim InputArray

    'Création d'un object de type ArrayList
    Set arr = CreateObject("System.Collections.ArrayList")

    'Array de chaîne de caracteres
    'InputArray = Array("d", "c", "b", "a", "f", "e", "g")

    'Array de nombres
    InputArray = Array(6, -3, 3, -5, 2, 1)

    'On passe les valeurs de l'Array de VBA a l'ArrayList de VBScript
    For Each element In InputArray
       arr.Add element
    Next

    'On trie
    arr.Sort

    'On reconvertit dans l'autre sens (ArrayList vers Array)
    sorted_array = arr.toarray
End Sub
```

Passer un array en argument est par contre un peu plus subtil que de passer une simple variable. Un exemple valant mieux que mille mots:

```
Sub Names()
    Dim arraySend(10) As Variant
    Dim arrayReceive As Variant

    arraySend(0) = "Weber"
    arraySend(1) = "Fournier"
    arraySend(2) = "Isoz"
    arrayReceive = maj(arraySend)
    MsgBox arrayReceive(0)
End Sub

Function maj(arraySend As Variant)
    For i = 0 To 2
        arraySend(i) = UCase(arraySend(i))
    Next i
    maj = arraySend
End Function
```

Nous voyons donc qu'un désavantage des tableaux en passage d'argument est que les données sont typées en Variant. Nous pouvons passer outre ce genre de limitation en jouant avec les **Type** comme le montre l'exemple ci-dessous:

```
Type TYP_Person
    nom As String
    age As Integer
    nb(2) As String
End Type

Sub Declaration()
    Dim typPerson As TYP_Person
    typPerson = Traitement()
    Debug.Print typPerson.nb(0)
End Sub

Function Traitement() As TYP_Person
    Traitement.nom = "Weber"
    Traitement.age = 10
    Traitement.nb(0) = 23
End Function
```


Dans la bibliothéque de l'explorateur d'objets vous pouvez finalement rencontrer les éléments suivants:

Les icônes de l'Explorateur d'objets			
Icône	Signification	Icône	Signification
	Type personnalisé On retrouve le même icône dans l'intellisense: équivaut à un type utilisateur: (User Defined): TYPE … END TYPE		*Propriété standard* Propriété par défaut: ex.: Label = "texte" équivaut à Label.caption="texte" N'existe plus en VB .NET
	Projet		Méthode

	Icône Projet		
	Classe		Méthode standard Méthode par défaut
	Module Icone Module		Enumération Ensemble de constants énumérée (voir dessous)
	Globale Membre appartenant à tous les sous membres en principe les constantes		Constante Valeur nommée: Ex: vbBlack = 0 Avantage si la valeur est modifiée le code continue à fonctionner
	Evénement		Propriété

8.1.8.2 Portée des variables

L'existence d'une variable peut se dérouler sur trois niveaux:

Niveau Procédure: cela veut dire que la variable est locale. Dès que l'on quitte la procédure en question, la variable disparaît, et son contenu avec elle. Pour déclarer une variable au niveau procédure, on tape à l'intérieur de la procédure:

```
Dim NomVariable as Type
```

Par exemple pour les variables les plus courantes nous pouvons voir les déclarations typiquement sous la forme suivante:

Déclaration Classique	Déclaration Court
Dim intEntier as Integer	Dim intEntier%
Dim lngEntier as Long	Dim lngEntier&
Dim sngNumeraire as Currency	Dim sngNumeraire@

Niveau Module: la variable est disponible pour toutes les procédures d'un Module, mais pas pour les procédures se situant sur un autre Module. Pour déclarer une variable au niveau Module, on tape tout en haut du Module, dans la partie (General):

```
Private NomVariable as Type
```

Une variable au niveau module est par défaut aussi une variable de type statique (static).

Niveau Projet: la variable est disponible, et sa valeur est conservée pour toutes les procédures de l'application, quel que soit leur emplacement. Pour déclarer une variable globale, il faut d'abord créer un module. Sur ce module, donc, on écrit:

.

```
Global NomVariable as Type
```

Naturellement, il ne faut pas raisonner en termes de facilité, et déclarer toutes les variables au niveau projet: car l'excès de place mémoire, ralentira votre application, au besoin considérablement. Il faut donc pour chaque variable se demander à quel niveau on en a besoin, et faire les bonnes déclarations en fonction.

L'existence d'une procédure peut se dérouler quant à elle que sur trois niveaux:

Niveau Module: une procédure privée ne pourra être invoquée que dans le module dans lequel elle est déclarée:

```
Private Sub NomProcédure()
   …
End Sub
```

Niveau Projet: une procédure publique peut-être invoquée de n'importe quel endroit du projet.

```
Public Sub NomProcédure()
```

```
    …
End Sub
```

L'existence des fonctions obéit aux mêmes règles, à la différence qu'il est possible en plus de spécifier le type de sortie de la variable:

```
Public Function NomFonction() as Type de données à renvoyer
    …
End Function

Private Function NomFonction() as Type de données à renvoyer
    …
End Function
```

Pour rendre obligatoire la déclaration de variables, placez l'instruction "`Option Explicit`" sur la première ligne du module ou cochez l'option "Déclaration des variables obligatoires" dans le menu **Outils/Options de l'éditeur de macros**.

Signolons que l'option:

```
Option Base 1
```

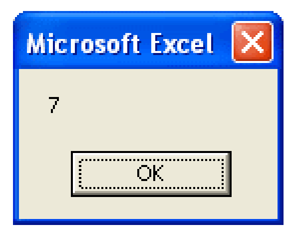
ou respectivement:

```
Option Base 0
```

permet de définir comment la numérotation des variables de type table doit débuter.

La déclaration explicite d'une variable se fait par le mot Dim (abréviation de Dimension). Le nombre maximum de caractères du nom de la variable est de 255. Il ne doit pas commencer par un chiffre et ne doit pas contenir d'espaces. La syntaxe est "Dim NomDeLaVariable as Type".

```
Sub Test()
    Dim SommeVal As Integer
    Dim Val1, Val2 As Integer
    'Evitez si possible de déclarer plusieurs variables sur une même ligne
car tous les premières seront en Variant excepté la dernière!!!!
    Val1 = 5
    Val2 = 2
    SommeVal = Val1 + Val2
    MsgBox Somme
End Sub
```



Vous pouvez également déclarer vos variables sur une même ligne:

```
Sub Test()
    Dim SommeVal As Integer, Val1 As Integer, Val2 As Integer
```

```
    Val1 = 5
    Val2 = 2
    SommeVal = Val1 + Val2
    MsgBox SommeVal
End Sub
```

La portée d'une variable est différente suivant l'endroit et la façon dont elle est déclarée.
Une variable déclarée à l'intérieur d'une procédure est dite "Locale". Elle peut-être déclarée par les mots Dim, Static ou Private. Dès que la procédure est terminée, la variable n'est plus chargée en mémoire sauf si elle est déclarée par le mot Static. Une variable Locale est généralement placée juste après la déclaration de la procédure.

```
Option Explicit
'Les variables Val1 et Val2 sont libérées de la mémoire alors que la variable SommeVal garde sa valeur à la fin de la procédure mais que dans la procédure même!
Public Sub Test()
    Static SommeVal As Integer
    Dim As Val1, Integer, Val2 As Integer
    'Instructions…
End Sub

Public Static Sub Test()
'Toutes les variables gardent leur valeur à la fin de procédure mais que dans la procédure même!
    Dim SommeVal As Integer
    Dim Val1 As Integer, Val2 As Integer
    'Instructions…
End Sub
```


Une variable peut être "Locale au module" si celle-ci est déclarée avant la première procédure d'un module. **Dès lors sa valeur sera conservée entre chaqu exécution de n'importe quelle routine du même module!** Toutes les procédures du module peuvent alors lui faire appel. Elle est déclarée par les mots Dim ou Private. Sa valeur sera

```
Option Explicit
'Les variables Val1 et Val2 peuvent être utilisées dans toutes les procédures du module
'attention!!! les variables publiques sont static par défaut
'il suffit d'écrire Static Public pour le voir car le mot Static va disparaître automatiquement
Public Val1 as  Integer, Val2 As Integer
'ou: Public/Private Val1 as  Integer, Val2 As Integer

Public Sub Test()

    Dim Val1 As Integer
    Dim Val2 As Integer
    'Essayez ensuite ce code en remplacent le mot Static par Dim
    Static SommeVal As Integer

    msgbox SommeVal
    SommeVal = Val1 + Val2
    msgbox SommeVal

End Sub
```

Un variable peut également être accessible à tous les modules d'un projet. On dit alors qu'elle est publique. Elle est déclarée par le mot Public. Elle ne peut pas être déclarée dans un module de Feuille ou dans un module de UserForm.

```
Option Explicit
'Les variables Val1 et Val2 peuvent être utilisées dans toutes les
procédures de tous les modules du projet.
Public Val1, Val2 As Integer
```

Une variable peut garder toujours la même valeur lors de l'exécution d'un programme. Dans ce cas, elle est déclarée par les mots Const ou Public Const.

```
Option Explicit
'La variable Chemin gardera toujours la valeur.
Const Chemin as String = "c:\application\excel\"
'Au besoin, pour rendre la constante globale, vous pouvez écrire Public
```

Il est possible de définir une taille fixe pour une variable de type String par la syntaxe Dim Variable as String * Longueur ou Longueur correspond au nombre de caractère que prend la variable.

```
Option Explicit

Public Sub Test

    Dim Couleur as String * 5
    Couleur = "Rouge"
    ' Si Couleur était égal à "Orange", la variable Couleur aurait pris
comme valeur "Orang".

End Sub
```


Vous pouvez également créer vos propres types de données à l'intérieur du bloc "Type-End Type" (déjà vu plus haut).

```
Option Explicit
'exemple de création d'un type de données personnalisé
Type Contacts
    Nom As String
    Prenom As String
    Age As Integer
End Type

Sub Test()

    'Déclaration de la variable du type personnalisé
    Dim AjoutContact As Contacts
    AjoutContact.Nom = "TOTO"
    AjoutContact.Prenom = "Titi"
    AjoutContact.Age = 20

End Sub
```

Les variables peuvent également faire référence à des objets comme des cellules, des feuilles de calcul, des graphiques, des classeurs ... Elles sont déclarées de la même façon qu'une variable normale.

```
Option Explicit

Public Sub Test()

    'La variable MaCel fait référence à une plage de cellules
    Dim MaCel As Range
    'Le mot Set lui affecte la cellule "A1"
    Set MaCel = Range("A1")
    'La cellule "A1" prend comme valeur 10
    MaCel.Value = 10
    'On détruit l'object
    Set MaCel = Nothing

    'Ou encore
    Dim wks as Object
    Set wks = ActiveWorkbook.Sheets(1)
    wks.Name = "Test"
    Set wks = Nothing

End Sub
```

Nous parlerons de la différence entre byVal et byRef plus tard dans les exercices pratiques.

8.1.8.3 Table des caractères ASCII

Table ASCII standard (codes de caractères de 0 … 127)

000	(nul)	016	(dle)	032	(sp)	048	0	064	@	080	P	096	`	112	p
001	(soh)	017	(dc1)	033	!	049	1	065	A	081	Q	097	a	113	q
002	(stx)	018	(dc2)	034	"	050	2	066	B	082	R	098	b	114	r
003	(etx)	019	(dc3)	035	#	051	3	067	C	083	S	099	c	115	s
004	(eot)	020	(dc4)	036	$	052	4	068	D	084	T	100	d	116	t
005	(enq)	021	(nak)	037	%	053	5	069	E	085	U	101	e	117	u
006	(ack)	022	(syn)	038	&	054	6	070	F	086	V	102	f	118	v
007	(bel)	023	(etb)	039	'	055	7	071	G	087	W	103	g	119	w
008	(bs)	024	(can)	040	(	056	8	072	H	088	X	104	h	120	x
009	(tab)	025	(em)	041	)	057	9	073	I	089	Y	105	i	121	y
010	(lf)	026	(eof)	042	*	058	:	074	J	090	Z	106	j	122	z
011	(vt)	027	(esc)	043	+	059	;	075	K	091	[	107	k	123	{
012	(np)	028	(fs)	044	,	060	<	076	L	092	\	108	l	124	|
013	(cr)	029	(gs)	045	-	061	=	077	M	093	]	109	m	125	}
014	(so)	030	(rs)	046	.	062	>	078	N	094	^	110	n	126	~
015	(si)	031	(us)	047	/	063	?	079	O	095	_	111	o	127	(127)

Table ASCII étendue (codes de caractères de 128 … 255)

128	€	144	□	160		176	°	192	À	208	Ð	224	à	240	ð
129	□	145	'	161	¡	177	±	193	Á	209	Ñ	225	á	241	ñ
130	,	146	'	162	¢	178	²	194	Â	210	Ò	226	â	242	ò
131	ƒ	147	"	163	£	179	³	195	Ã	211	Ó	227	ã	243	ó
132	„	148	"	164	¤	180	´	196	Ä	212	Ô	228	ä	244	ô
133	…	149	•	165	¥	181	µ	197	Å	213	Õ	229	å	245	õ
134	†	150	–	166	¦	182	¶	198	Æ	214	Ö	230	æ	246	ö
135	‡	151	—	167	§	183	·	199	Ç	215	×	231	ç	247	÷
136	ˆ	152	˜	168	¨	184	¸	200	È	216	Ø	232	è	248	ø
137	‰	153	™	169	©	185	¹	201	É	217	Ù	233	é	249	ù
138	Š	154	š	170	ª	186	º	202	Ê	218	Ú	234	ê	250	ú
139	‹	155	›	171	"	187	"	203	Ë	219	Û	235	ë	251	û
140	Œ	156	œ	172	¬	188	¼	204	Ì	220	Ü	236	ì	252	ü
141	□	157	□	173		189	½	205	Í	221	Ý	237	í	253	ý
142	Ž	158	ž	174	®	190	¾	206	Î	222	Þ	238	î	254	þ
143	□	159	Ÿ	175	¯	191	¿	207	Ï	223	ß	239	ï	255	

8.1.8.4 Nomenclature de Lezsynski/Reddick

Pour le développement (base de données et autres), il y a certaines règles et traditions qu'il vous faut respecter dans un premier temps pour votre confort et dans un deuxième temps pour être compatible avec vos collègues et les possibles futures migrations.

Les règles Lezsynski/Reddick© pour le développement de base de données sont les suivantes (elles ont été également adoptées pour d'autres langages de programmation):

Majuscule au début de chaque mot d'une variable, pas d'accents, pas de caractères spéciaux, pas d'espaces, nom des champs en anglais, éviter de dépasser les 8 caractères, ne jamais commencer avec des chiffres:

- Nom des tables: tbl….
- Nom des requêtes: qry…
- Nom des vues: vue…
- Nom des états: rpt…
- Nom des formulaires: frm…
- Nom des champs clés primaire avec numéro automatique: idNomTable
- Nom des champs clés étrangères: tblNomTableNomChamp
- Nom de tous les autres champs:

strNom…, intNom…, datNom…, oleNom…, hypNom…, bolNom…

- Nom des champs de formulaire:

lstNomListe…., optGroupeOptions…., chkChoixCase…., tglToggleButton…, fldNomChamp…

Exemple d'une variable: *intStreetNb.* Ce qui est beaucoup mieux que *Numéro de la rue* qui ne nous donne pas d'un premier coup d'œil, le type de données dont il s'agit, qui n'est pas compatible avec la quasi-totalité des langages de programmation, et qui peut être compris par un maximum de personne de par l'usage de la langue anglaise.

Il est aussi possible d'utiliser une version condensée de la norme ci-dessous connue sour le nom "syntaxe Camel" utilisée par la majorité des développeurs (seniors).

Préfixes de variables:

Préfixe	Emploi de la variable	Exemple de variable
b ou bln	Booléen	blnSuccess
c ou cur	Monnaie	curAmount
d ou dbl	Double	dblQuantity
dt ou dat	Date et heure	datDate
f ou flt	Flottant	fltRatio
l ou lng	Long	lngMilliseconds
i ou int	Entier	intCounter
s ou str	Chaîne	strName
a ou arr	Tableau	arrUsers()
o ou obj	Objet COM	objPipeline
d ou dec	Decimal	decSomme
byt	Byte	bytNumRue

Préfixes de variables pour les objets de base de données:

Préfixe	Emploi de la variable	Exemple de variable
cnn	Connexion	cnnPubs
rst	Jeu d'enregistrements	rstAuthors
cmd	Commande	cmdEmployee
fld	Champ	fldLastName
frm	Formulaire	frmHome
mod	Module	modProcedures
prj	Projet	prjFormation
cls	Module de classe	clsBoutons

Préfixes d'étendue et d'usage:

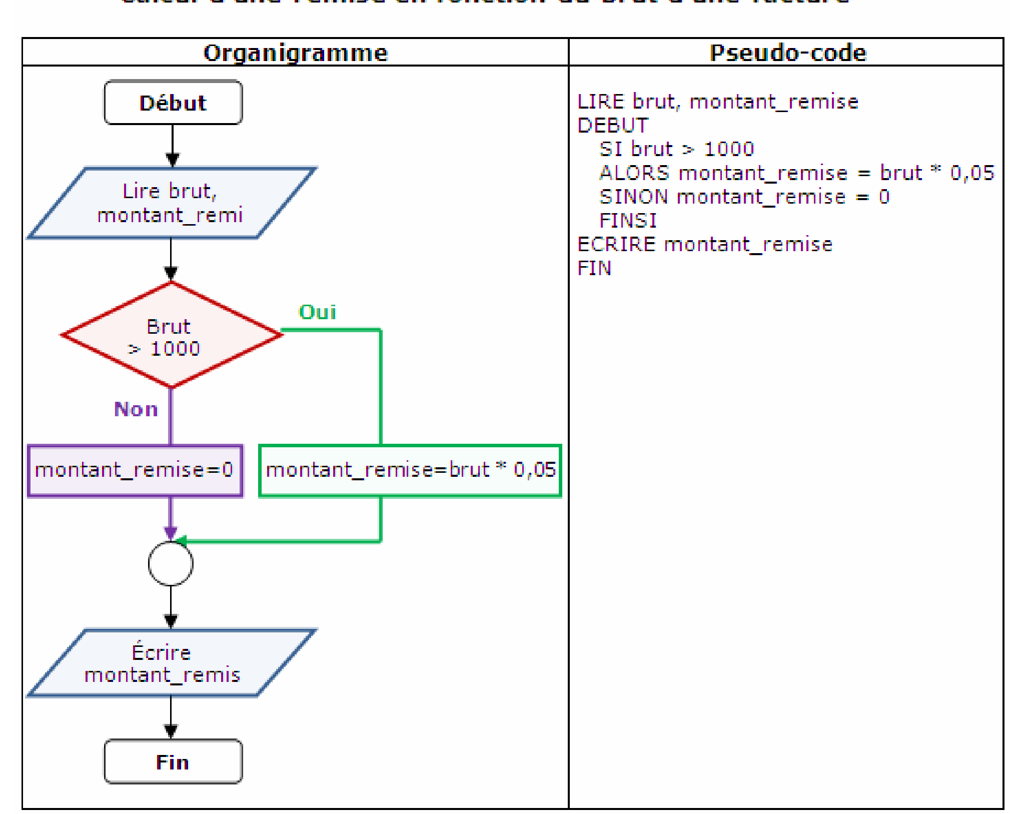
Préfixe	Description
g_	Usage Public
m_	Usage Local
(pas de préfixe)	Variable non statique, préfixe local à la procédure

Les six règles d'or d'usage:

R1. Toujours en anglais
R2. Entre 8 et 11 caractères
R3. Pas de caractères spéciaux
R4. Pas d'espaces
R5. Toujours les 3 premières lettres du type de données
R6. Une majuscule à chaque premier lettre des mots des variables

8.1.9 Structures conditionnelles

Il existe deux types de structures conditionnelles en V.B.A. (comme dans la majorité des languages de programmation). On peut toujours utiliser l'un ou l'autre à ma connaissance mais les développeurs ont des préférences d'écriture qui s'imposent d'elles-mêmes en termes de facilité de relecture du code.



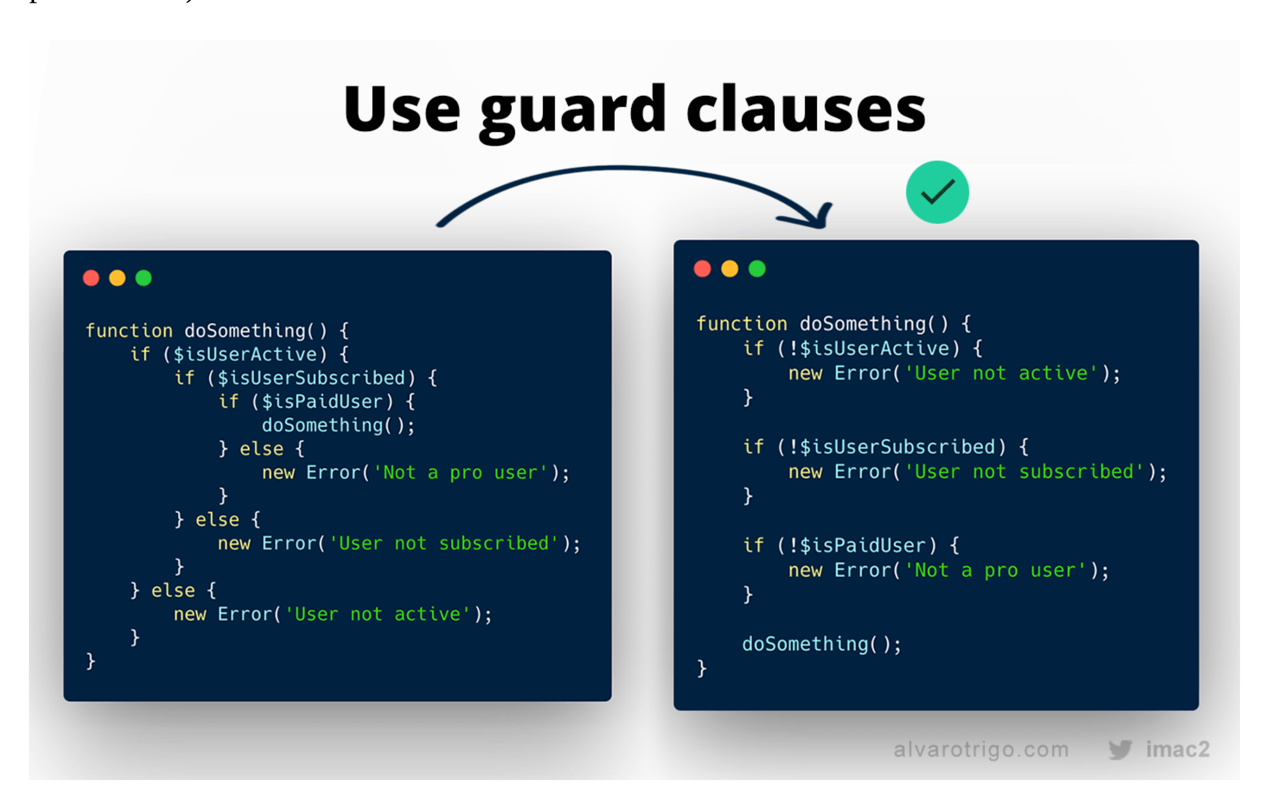
Ces deux types de structures conditionnelles sont IF et SELECT. En général dans les applications financières et d'ingénierie demandant une grande rapidité on préféra le SELECT qui est statistiquement plus rapide que IF de 5% à 20% dans la majorité des cas.

Le lecteur pourra vérifier avec le code suivant en exécutant tantôt le premier, tantôt le deuxième plusieurs fois de suite:

```
Sub testIf()
    Dim t As Single
    t = Timer
    For i = 1 To 1000000 Step i
        If Cells(i, 1) > 0 Then
            Debug.Print "+1"
        ElseIf Cells(i, 1) < 0 Then
            Debug.Print "-1"
        Else
            Debug.Print 0
        End If
    Next i
    MsgBox Timer - t
End Sub
```

```
Sub testCase()
    Dim t As Single
    t = Timer
    For i = 1 To 1000000 Step i
        Select Case Cells(i, 1)
        Case Is > 0
            Debug.Print "+1"
        Case Is < 0
            Debug.Print "-1"
        Case Else
            Debug.Print 0
        End Select
    Next i
    MsgBox Timer - t
End Sub
```

Attention! Certains professionnels de la programmation utilisent la technique de la "Guard Clause" qui permet de rendre le code beaucoup plus lisible plutôt que d'imbrique des IF… ELSE les uns dans les autres! La technique peut être illustrée par la figure suivante (qui n'est pas en VBA):

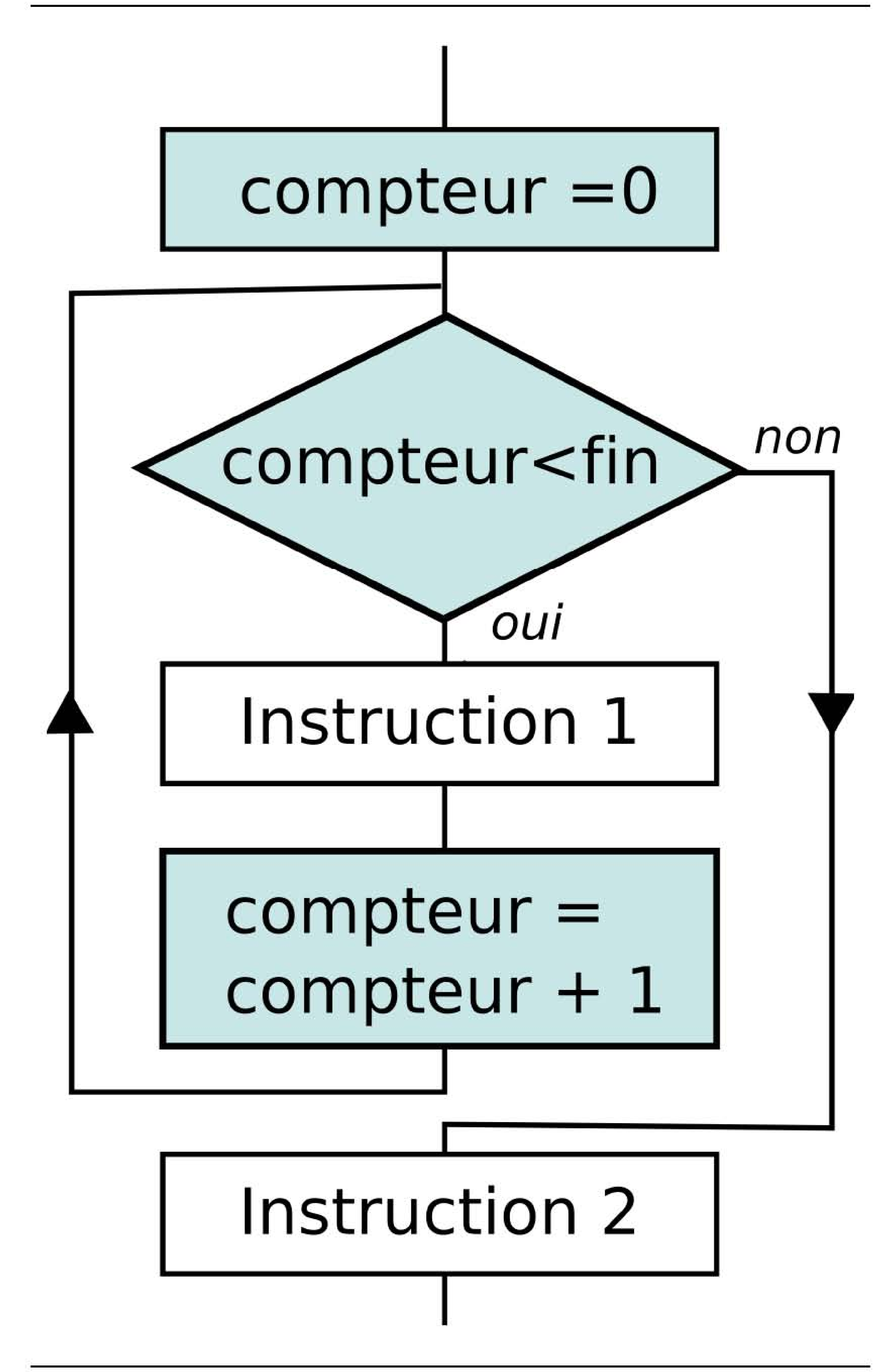


8.1.10 Structure itératives

En informatique, la structure itérative est une structure de contrôle de programmation qui permet de répéter l'exécution d'une séquence d'instructions.

Selon les langages de programmation, différents mots-clés sont utilisés pour signaler cette structure de contrôle : **for** pour les descendants d'Algol, **do** pour FORTRAN, PL/I, etc.

Par exemple "boucle for" a deux parties : une entête qui spécifie la manière de faire l'itération, et un corps qui est exécuté à chaque itération:


compteur =0
compteur<fin
non
oui
Instruction 1
compteur =
compteur + 1
Instruction 2

```
'*******************************
'Créateur: Vincent ISOZ
'Dernière modification: 28.10.2003
'Nom fonction: factitfor()
'Commentaires: Calcul de la factorielle d'un nombre n par la méthode
itérative "for"
'Objectif de cours: apprendre a créer des fonction itératives
'*******************************

Public Function factitfor(n As Integer)

'On ne déclare pas la variable factit qui a le même nom que la fonction!!
Dim I As Integer

    factitfor = 1
    For I = 1 To n
        factitfor = factitfor * i
    Next i

End Function
```

Attention!!! Sauter une itération est un peu particulier en VBA (particulièrement moche). Voyons un exemple bête et simple sans application pratique, juste pour voir la syntaxe…

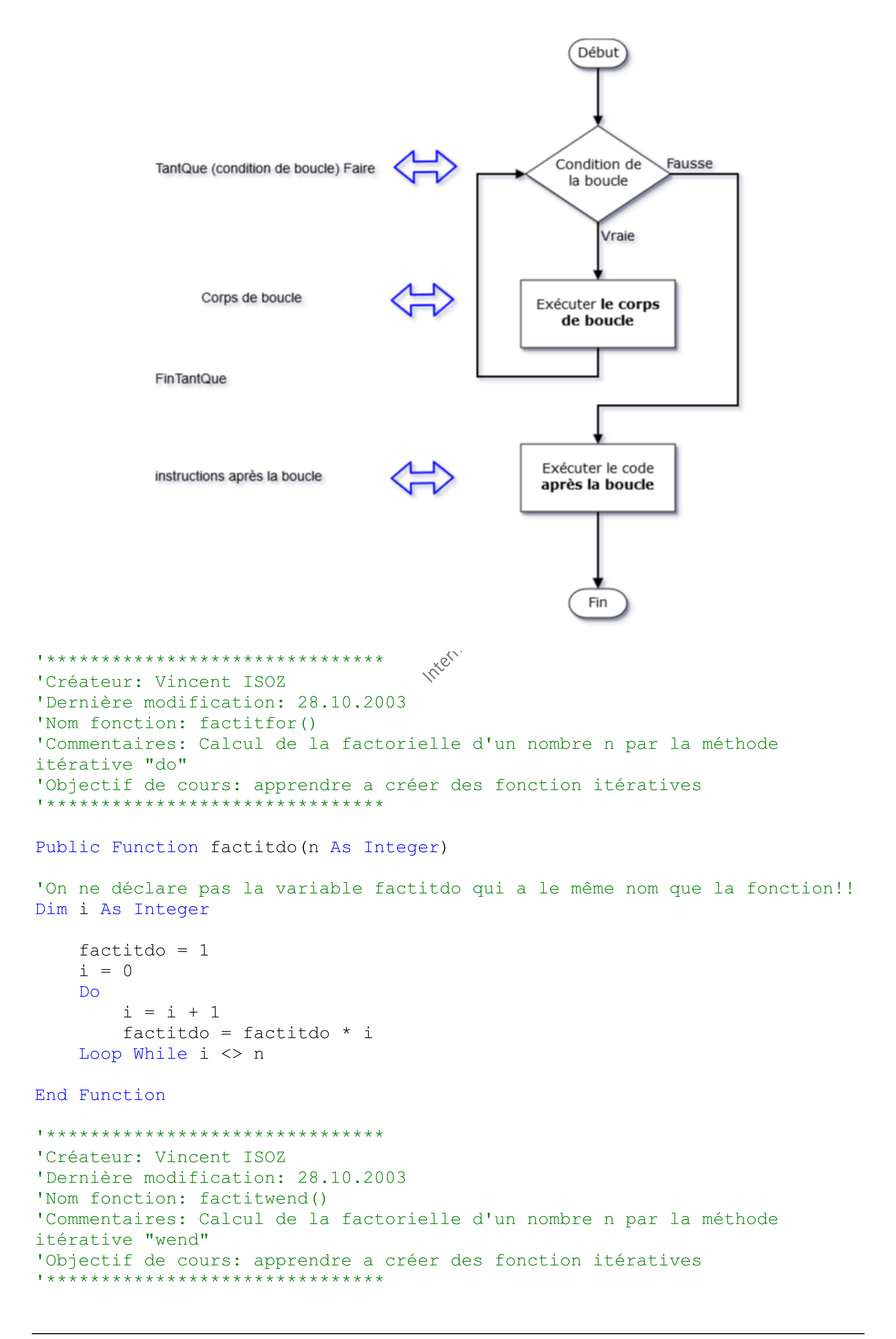
```
Sub BoucleForAvecSaut

Dim I As Integer

    I = 10
    For I = 1 To 10
        Debug.Print I
        If I = 5 Then
           GoTo SauteIteration
        End If
SauteIteration:
    Next i

End Function
```


Il existe cependant d'autres structures de boucles basées sur des conditions plutôt que sur un compteur! Le concept est résumé par le flux suivant:




```
'*******************************
'Créateur: Vincent ISOZ
'Dernière modification: 28.10.2003
'Nom fonction: factitfor()
'Commentaires: Calcul de la factorielle d'un nombre n par la méthode
itérative "do"
'Objectif de cours: apprendre a créer des fonction itératives
'*******************************

Public Function factitdo(n As Integer)

'On ne déclare pas la variable factitdo qui a le même nom que la fonction!!
Dim i As Integer

    factitdo = 1
    i = 0
    Do
        i = i + 1
        factitdo = factitdo * i
    Loop While i <> n

End Function

'*******************************
'Créateur: Vincent ISOZ
'Dernière modification: 28.10.2003
'Nom fonction: factitwend()
'Commentaires: Calcul de la factorielle d'un nombre n par la méthode
itérative "wend"
'Objectif de cours: apprendre a créer des fonction itératives
'*******************************
```

```
Public Function factitwend(n As Integer)

'On ne déclare pas la variable factitwend qui a le même nom que la
fonction!!
Dim i As Integer

    factitwend = 1
    i = 1
    While i<=n
        factitwend = factitwend * i
        i = i + 1
    Wend

End Function
```

8.1.11 Commentaires VBA

Les commentaires du code est un point très important du développement que l'on comprend seulement avec l'expérience. Effectivement, les développeurs débutants n'ont dans un premier temps pas l'habitude de travailler très rigoureusement et très proprement et ce d'autant plus sur des codes rarement supérieurs à 1000 lignes et ne voient donc pas quelle est l'intérêt futur de bien commenter leur code. Cet état des faits a lieu chez la grande majorité des développeurs.

Ne pas commenter est une énorme erreur pour le développeur lui-même et tous ceux qui seraient amenés à intervenir ou à poursuivre son travail.

Certaines règles sont à mettre en place il convient immédiatement de mettre en pratique dès que l'on commence à rédiger un code. Voici ces règles:

R1. Toute procédure, fonction, classe, doit être accompagnée d'une cartouche de description telle que dans l'exemple ci-dessous

R2. Chaque ligne de code doit être commentée avec indication en initiales du commentateur et de la date de création du commentaire tel que dans l'exemple ci-dessous

R3. Au besoin, un schéma procédural doit être fait dans un logiciel adapté (MS Visio pour V.B.A. suffit) pendant le travail afin de savoir qui appelle quoi en faisant usage de quelles variables

Exemple de code:

```
'********************************
'Créateur(s): Vincent ISOZ
'Dernière modification: 18.09.2004
'Nom fonction: TestDeVariable()
'Appelée par: -
'Commentaires: exemple de danger de conversion de données
'********************************

Dim sng As Single 'Nombre réel simple précision
Dim dbl As Double 'Nombre réel double précision

Public Sub SigngleToDouble()

     'on affecte 1.9 a la variable sng
     sng = 1.9
     'on affecte la valeur de sng a dbl
     dbl = sng
     'on Affiche dbl
     MsgBox dbl
     'ou encore pour les sceptiques
     dbl=Cdbl(sng)
     MsgBox dbl

End Sub

'********************************
'Créateur(s): Vincent ISOZ
'Dernière modification: 28.10.2003
'Nom fonction: factitfor()
'Appelée par: mettre ici les noms de procédures (avec les modules) qui
appellent la fonction
'Appelle: mettre ici le nom des de procédures (avec les modules) qui sont
appelé par la fonction
'Commentaires: Calcul de la factorielle d'un nombre n par la méthode
itérative "for"
'Objectif de cours: apprendre a créer des fonction itératives
'********************************

Public Function factitfor(n)

'V.I.(28.10.03): On ne déclare pas la variable factit qui a le même nom que
la fonction!!
Dim i As Integer
```

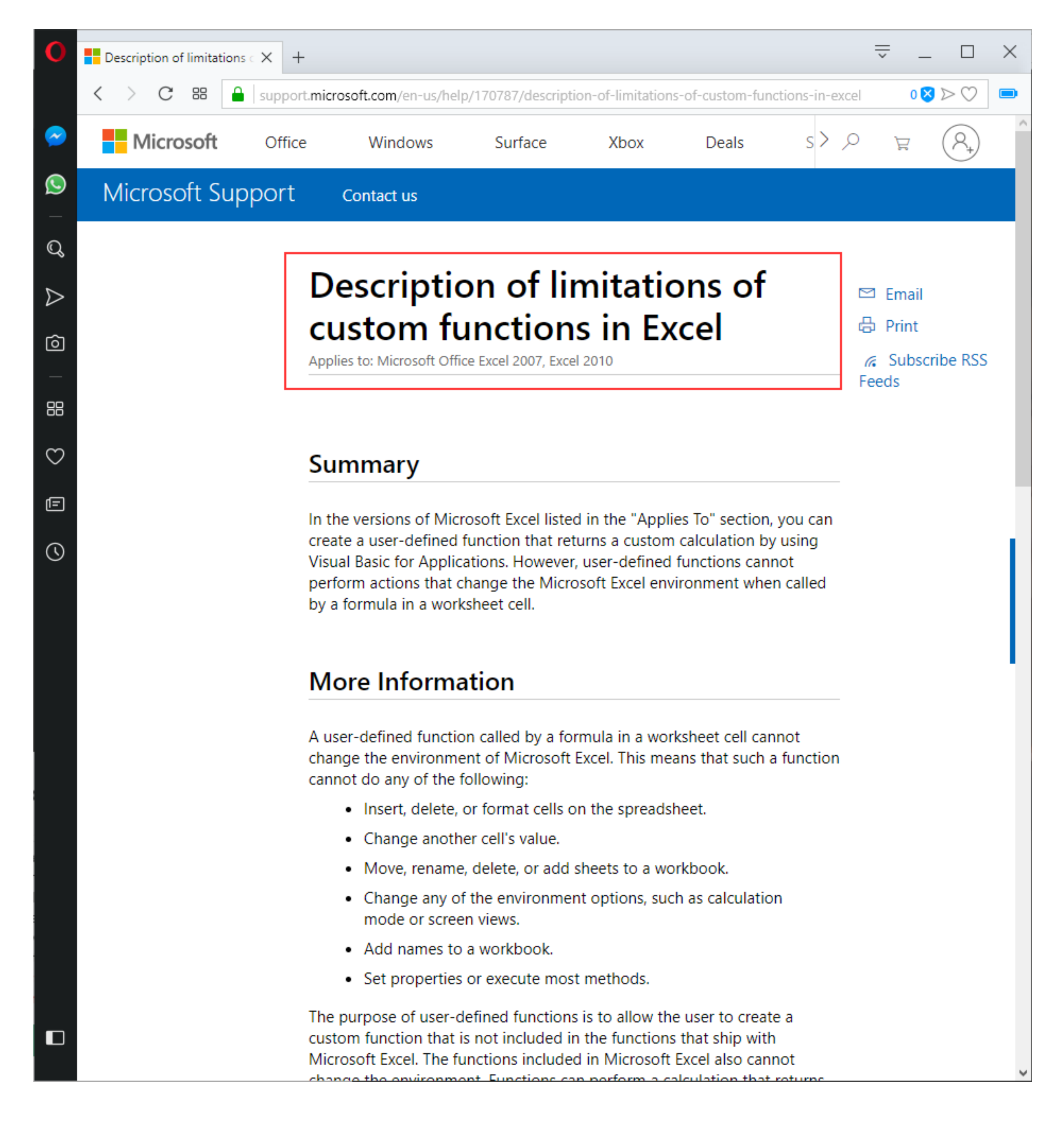
```
    factitfor = 1
    'V.I.(28.10.03): On utilise la méthode itérative classique vue à
l'école primaire
    'Attention, n et i doivent être des variables du même type !
    For i = 1 To n
        factitfor = factitfor * i
    Next i

End Function
```

9. FONCTIONS (exemples)

Les fonctions V.B.A. sont normalement des ensembles de codes qui retournent des valeurs mais n'effectuent pas d'actions. Il faut savoir que cependant qu'il est possible de faire de même avec des procédures (excepté pour MS Excel – sauf contournement relativement très techniques - qui constitue un cas à part dans la suite MS Office car les fonctions y ont un statut particulier). C'est pour cette raison que dans certains langages modernes les fonctions n'existent plus.

Voici une capture d'écrand la page Internent officielle de Microsoft concernant la limitation des fonctions avec Microsoft Excel (la partie importante est encadrée en rouge):



Quelques petits exemples de fonctions avant de se faire la main sur les procédures. Certaines fonctions seront ré-utilisées plus tard dans les Userforms et dans les procédures.

```
'Oblige le programmeur à déclarer rigoureusement et proprement ses variables
Option Explicit

'!!!!!!!!!!!!!!!!!!
'Remarque: la syntaxe des variables présentée ici ne satisfait
'pas la norme habituelle par souci de clarté.
'Le formateur approfondira et argumentera le sujet
'La philosophie de développement est similaires pour tous les
'logiciels de la suite MS Office System©
'!!!!!!!!!!!!!!!!!!
```

Voici les principes de bases de déclaration des fonctions:

```
' Cette fonction définie par l'utilisateur renvoie la
' racine carrée de l'argument qui lui est transmis.

Public Function CalculateSquareRoot(NumberArg As Double) As Double
   'la rigueur voudrait que chaque fois après la fonction, nous rédéfinissions le type de
   variable renvoyée. Cependant,  c'est souvent un dérivé de l'argument qui est renvoyé donc
   cette écriture est un peu redondante (testez en changeant ci-dessus Double comme un
   String et testez également pour CalculateSquareRoot – avec IsDouble – de quel type de
   donné il s'agit).
   If NumberArg < 0 Then
      Exit Function
   Else
      ' Renvoie la racine carrée.
      CalculateSquareRoot = Sqr(NumberArg)
   End If
End Function
```


On peut ranger cette function dans une des catégories de fonctions à l'aide du code suivant (avant de transformer le fichier contenant les fonctions en un *.xla):

```
Private Sub Workbook_AddinInstall()
    Application.MacroOptions Macro:="CalculateSquareRoot", Description:="Calcule la
    racine carré d'une valeur réelle positive", Category:="Finances"
End Sub
```

Le mot clé ParamArray permet à une fonction d'accepter un nombre variable d'arguments. Dans la définition suivante, FirstArg est transmis par valeur.

```
Public Function CalcSum(ByVal FirstArg As Integer, ParamArray OtherArgs())
Dim ReturnValue
….
```

Si la fonction est appelée sous la forme: `CalcSum(4, 3 ,2 ,1)`, les variables locales ont les valeurs suivantes: `FirstArg = 4, OtherArgs(1) = 3, OtherArgs(2) = 2`, etc., en supposant que la limite inférieure par défaut des tableaux = 1.

Les arguments `Optional` peuvent être dotés de types et de valeurs par défaut autres que Variant si les arguments d'une fonction sont définis sous la forme:

```
Function MyFunc(MyStr As String, Optional MyArg1 As Integer = 5, Optional
MyArg2 = "Dolly")
….
```

Si nous voulons pouvoir utiliser la propriété `isMissing` afin de vérifier qu'un paramètre optionnel a bien été passé, il est obligé de le mettre en `Variant`:

```
Function User(Optional UpperCase As Variant)
   If IsMissing(UpperCase) Then UpperCase = False
   User = Application.UserName
   If UpperCase Then User = UCase(User)
End Function
```

La fonction peut dès lors être appelée de plusieurs façons:

1. Les 3 arguments sont fournis: `RetVal = MyFunc("Bonjour", 2, "à tous")`
2. Le deuxième argument est omis: `RetVal = MyFunc("Test", , 5)`
3. Arguments un et trois indiqués en utilisant des arguments nommés: `RetVal = MyFunc(MyStr:="Bonjour ", MyArg1:=7)`

```
'Attention!!!
'Si vous écrivez "Option Explicit" sur le module vous serez obligé de
déclarer toute variable!!!Sinon quoi elles sont gérées comme des "Variant"
(conseillé de le déclarer par rigueur)
```

```
'********************************
'Créateur: Vincent ISOZ
'Dernière modification: 28.10.2003
'Nom fonction: utilisateur()
'Commentaires: Renvoie le nom de l'utilisateur dans la cellule ou la
fonction est écrite
'et écrit dans une boite de message le nom inversé
'Objectif du cours: apprendres les commandes "UCase", "MsgBox", "TextLen",
"Beep", "Mid"
'et la récursivité, commandes que l'on retrouve dans les autres logiciels
de la suite office
'********************************

Public Function utilisateur()

Dim textlen, renverse, utilisateur, entreprise As String
Dim i As Integer

    utilisateur = Application.UserName
    'On met en majuscules le UserName
    utilisateur = UCase(utilisateur)
    entreprise = ActiveWorkbook.BuiltInDocumentProperties("Company")
    'On affiche le tout dans une message box (voi l'aide!! pour le retour
chariot par exemple)
    MsgBox "Excel est enregistré sous: " & utilisateur & " (" & entreprise
& ")" , vbOKOnly + vbInformation, Enregistrement
    'Rigoureusement il faudrait écrire … Interaction.MsgBox mais bon…
    'On compte combien de lettres il y a dans le nom de l'utilisateur
    textlen = Len(utilisateur)
    'Première structure de boucle de type For
    'Par pas de 1 on analyse en reculant les lettres du nom de
l'utilisateur
    For i = textlen To 1 Step -1
        'On émet un son à chaque boucle
        Beep
        'Vous n'êtes pas obligés de choisir i comme variable d'itération
        'On parcourt 1 par 1 les caractères et on les concatène avec le
caractère précédent
        'les troisième argument de la fonction Mid donne le nombre de
caractères à retourner
        renverse = renverse & Mid(utilisateur, i, 1)
        MsgBox "Etapes du mot renvérsé" & VbCrlf & renverse
    Next i
    utilisateur = renverse
    'Une fois la fonction terminée et testée allez voir dans les fonctions
d'Excel
    'sous la catégorie "Personnalisée":-)

End Function
```


Quelques infos:

Constante	Caractère	
vbCrlf	Retour à la ligne	Pour certaines situations (car cela ne marche pas pour tous les objets)
chr(10)	Retour à la ligne	Pour certaines situations (car cela ne marche pas

	pour tous les objets)
Il faut parfois jouer entre vbCrlf et chr(10) pour certains objets.	

Constante	Icône	
vbCritical		Pour une erreur fatale
vbExclamation		Pour une remarque
vbInformation		Pour une information
vbQuestion		Pour une question

Constante	Boutons		
vbAbortRetryIgnore	Abandonner	Recommencer	Ignorer
vbOKCancel	OK	Annuler	
vbRetryCancel	Recommencer	Annuler	
vbYesNo	Oui	Non	
vbYesNoCancel	Oui	Non	Annuler

Constante	Valeur
vbOK	1
vbCancel	2
vbAbort	3
vbRetry	4
vbIgnore	5
vbYes	6
vbNo	7

```
'*******************************
'Créateur: Vincent ISOZ
'Dernière modification: 28.10.2003
'Nom fonction: montotal()
'Commentaires: Cette fonction calcule un simple produit sur la base d'un
prix et d'une taxe
'Objectif de cours: apprendre a passer plusieurs paramètres et les
strucutres conditionnelles
'*******************************

Public Function MonTotal(produit As Integer, tva As Single)

    If tva = 0 Then
        MsgBox "Vous devez entrer une valeur pour la TVA compris entre
[5;10]%"
    ElseIf tva < 0.05 And tva > 0 Then
        MsgBox "La valeur de la TVA est trop faible"
    ElseIf tva > 0.1 Then
        MsgBox "La valeur de la TVA est trop grande"
    Else
        MonTotal = produit * tva + produit
    End If

End Function


'*******************************
'Créateur: Vincent ISOZ
'Dernière modification: 28.10.2003
'Nom fonction: montotal()
'Commentaires: Cette fonction retourne le nombre de voyelles d'une suite de
mots
'Objectif de cours: apprendre à utiliser les commandes traitant sur les
textes
'et les commandes de débogage
'*******************************

Public Function CompteVoyelles(texte As String)

Dim compte As Integer
Dim ch As String

    'On initialise une variable (ce qui n'est pas tjrs) obligatoire
    compte = 0
    'On va compter les voyelles
    For i = 1 To Len(texte) 'à comparer avec l'exercice précédent...
        ch = Mid(texte, i, 1)
        'on teste la caractère pour voir si c'est une variable
        'Regardez la syntaxe de la commande Like dans l'aide
        'Regardez la syntaxe de la commande Like dans l'aide
        If ch Like "[aeiou]" Then
            compte = compte + 1
            'on affiche le résultat intermédiaire dans la fenêtre
d'exécution
            Debug.Print ch, i
        End If
    Next i
    'Ecrivez la fonction dans une feuille et appelez dans l'argument une
cellule contenant un texte
```

```
End Function
'********************************
'Créateur: Vincent ISOZ
'Dernière modification: 28.10.2003
'Nom fonction: suprespaces()
'Commentaires: Cette fonction supprime les espaces contenus dans un texte
'Objectif de cours: apprendre à utiliser les commandes traitant sur les
textes
'et les commandes de débogage (suite...)
'********************************

Public Function suprespaces(texte As String)

    Dim Temp, ch as String

    For i = 1 To Len(texte)
        ch = Mid(texte, i, 1)
        '32 est la valeur ascii du l'espace vide
        If ch <> Chr(32) Then
            Temp = Temp & ch
        End If
    Next i
    suprespaces = Temp
    'Si vous connaissiez bien les instuctions V.B.A. le contenu ci-dessus
aurait pu s'abréger
    MsgBox Replace(texte, " ", "")

End Function

'********************************
'Créateur: Vincent ISOZ
'Dernière modification: 28.10.2003
'Nom fonction: afficheascii()
'Commentaires: Cette fonction affiche la table ascii des caractères
'33 à 126 (valeurs hardcodées). La connaissance des caractères ASCII
'est souvent utile en VBA
'********************************

Public Function AfficheAscii()
Dim i As Integer
Dim debutascii, finascii As Integer
debutascii = 33
finascii = 126

    For i = debutascii To finascii
        Debug.Print i, Chr(i)
        'Tapez dans le débugueur:
        'x=2*2
        'msgbox x
        'ou encore
        'msgbox replace("hallo","a","e")
    Next i

End Function
```

```
'*******************************
'Créateur: Vincent ISOZ
'Dernière modification: 28.10.2003
'Nom fonction: ajmois()
'Commentaires: Cette fonction ajoute des un valeur donnée
'à l'année, mois ou jour d'une date (sans choix possible mais cela...
'peut se faire comme exercice)
'Objectif de cours: apprendre à manipuler la principale commande relative
aux dates
'*******************************

Public Function ajmois(datedepart As Variant, delaimens As Variant)

    'on contrôle si l'argument datedepart est bien du format de type 'date'
    If IsDate(datedepart) = False Or IsNumeric(delaimens) = False Then
    'nous pourrions écrire: not(isdate(datedepart)) or
not(isnumeric(delaimens))
        'On affiche une boîte de dialogue s'il y a une erreur dans
l'argument
        MsgBox "Date entrée ou délai mensuel invalide", vbCritical
        'et on sort de la fonction car on ne peut continuer
        Exit Function
    End If
    'Pour convertir la date et avoir le mois en toutes lettres
    'MonthName(Month(Date), False)
    'Mais si l'argument est au bon format
    ajmois = DateSerial(Year(datedepart), Month(datedepart) + delaimens,
Day(datedepart))
    'allez voir l'aide pour les autres formats que 'short date'

End Function

'*******************************
'Créateur: Vincent ISOZ
'Dernière modification: 28.10.2003
'Nom fonction: divisedeux()
'Commentaires: Cette fonction divise deux nombres entre eux
'Objectif de cours: apprendre à gérer correctement les erreurs
'Exercice: compléter les fonctions précédentes au mieux
'*******************************

Public Function DiviseDeux(nb1 As Variant, nb2 As Variant)

    'Nous mettons en Variant afin de laisser passer les lettres au delà de
la fonction

    If IsNumeric(nb1) = False Or IsNumeric(nb2) = False Then
    'a nouveau nous pourrions écrire: not(isnumeric(nb1)) or
not(isnumeric(nb2))
        MsgBox "Une des deux entrées n'est pas un nombre", vbCritical +
vbRetryCancel, "Attention!"
    End If

On Error GoTo GestionErreurs
    divisedeux = nb1 / nb2
    'N'oubliez pas de quitter la fonction sinon quoi la gestion des erreurs
va être executée
    Exit Function
    'Ou Exit Sub dans le cas d'une routine
```

```
GestionErreurs:
    MsgBox Str(Err.Number) & ": " & Err.Description, , "Erreur"
    'le message à renvoyer dans la cellule en cas d'erreur
    nb2=inputbox("Nouvelle valeur pour nb2")
    'Vous pouvez écrire ensuite (pour continuer avec autre chose):
    'Call NomAutreProcedureAExecuter
    'ou
    Resume 'revient à la ligne qui a provoqué l'erreur

End Function

'********************************
'Créateur: Vincent ISOZ
'Dernière modification: 28.10.2003
'Nom fonction: lignedumax()
'Commentaires: Cette fonction renvoie la valeur maximale d'une colonne
'dont on a spécifié le numéro au préléable
'Objectif de cours: apprendre à lire le contenu d'une cellule
'Exercice: compléter la fonction au niveau de la gestion des erreurs
'********************************

Public Function lignedumax(col As Integer)

Dim nblignes, valmax As Double
Dim i As Integer

    'On compte le nombre de lignes qu'il y a au total dans la colonne
    nblignes = Rows.Count
    'On recherche la valeur maximale
    valmax = Application.WorksheetFunction.Max(Columns(col))
    'On va parcourir les lignes 1 par 1 jusqu'à ce qu'on trouve la valeur
max
    For i = 1 To nblignes
        'Si on trouve la valeur Max
        If Cells(i, col) = valmax Then
            lignedumax = i
            'On sort de la boucle car on a trouvé
            Exit For
        End If
    Next

End Function

'********************************
'Créateur: Vincent ISOZ
'Dernière modification: 28.10.2003
'Nom fonction: lirecellule()
'Commentaires: a pour object de montrer presque toutes les différentes
méthodes permettant d'accéder au cotenu d'une cellule
'Objectif de cours: accès aux données dans une feuille
'********************************

Public Function lirecellule()

    'On affiche dans une boîte de dialogue le contenu de la cellule A1
    'Remarque: une fonction ne peut pas écrire une donnée ailleurs que
    'dans la cellule où elle a été saisie !!

    MsgBox Range("A1").Value
    MsgBox Cells(1, 1) 'et attention il ne faut pas écrire
Cells(1,1).Value!!!
```

```
    MsgBox Cells(1)
    'On affiche dans une autre boîte de dialogue le nombre total de
cellules d'une feuille excel
    MsgBox Range("a1:iv65536").Count
    lirecellule = "Pas mal non?"

End Function

'*******************************
'Créateur: Vincent ISOZ
'Dernière modification: 29.11.2003
'Nom fonction: pascal ()
'Commentaires: cette procédure génère le non moins fameux "triangle de
pascal"
'Objectif de cours: écrire dans des cellules, utilisation des fonctions
incluses dans MS Excel
'Attention il existe deux manières différentes d'écrire le Until, une avec
la condition après le Do une avec la condition écrite après le Loop.
'*******************************

Public Sub Pascal()

    Dim i, n As Double

    n = InputBox("limite du triangle")
    n = n + 1
    i = 1
    Do Until i = n
        Cells(i, 1) = 1
        i = i + 1
    Loop
    n = 1
    Do Until n = 10
        n = n + 1
        p = 1
        Do Until p = n
             Cells(n, p + 1) = WorksheetFunction.Combin(n, p)
             p = p + 1
        Loop
        Cells(n, p + 1) = 1
    Loop

End Sub

'*******************************
'Créateur: Vincent ISOZ
'Dernière modification: 28.10.2003
'Nom fonction: sumcompmant()
'Commentaires: Cet exercice a pour objectif d'apprendre
'à utiliser les cellules d'une plage de données pour des calculs
'Objectif de cours: utilisation des matrices, plages de cellules (plages
contigues!!!)
'*******************************

Public  Function SumCompMatR1(Plage As Range)
    'Cette fonction ne marche que sur des sélections contigues
    'On ne déclare pas le type de tableau car cela est déjà fait avec
"Plage"
    Dim Tableau
    Dim Cumul As Double
    Dim i,j As Integer
```

```
    Tableau = Plage

    'La commande UBound permet de connaître la dimension du tableau en long
et en large
    'On souhaite d'abord connaître le nombre de colonne d'où le "1" et le
nombre de lignes "2"
    MsgBox "Il y a " & UBound(Tableau, 1) & " lignes et " & UBound(Tableau,
2) & " colonnes"
    For i = 1 To UBound(Tableau, 1)
        For J = 1 To UBound(Tableau, 2)
            'On affiche juste le contenu du tableau pour voir si tout
fonctionne
            MsgBox Tableau(i, J)
            Cumul = Cumul + Tableau(i, J)
            'On affiche le cumul pour voir si tout fonctionne comme prévu
            MsgBox "Le cumul est " & Cumul
         Next j
    Next i
    sumcompmatR1 = Cumul

End Function

'********************************
'Créateur:?
'Dernière modification: 17.05.2011
'Nom fonction: ColumLetter
'Commentaires: Convertir chiffre en lettre de colonne de feuille Excel
'********************************
Public Function ColumnLetter(ByVal ColumnNumber As Integer) As String
  If ColumnNumber > 26 Then
    ColumnLetter = Chr(Int((ColumnNumber - 1) / 26) + 64) & _
                   Chr(((ColumnNumber - 1) Mod 26) + 65)
  Else
    'Columns A-Z
    ColumnLetter = Chr(ColumnNumber + 64)
  End If
End Function

'********************************
'Créateur: Vincent ISOZ
'Dernière modification: 28.10.2003
'Nom fonction: sumcompmantR2()
'Commentaires: même chose qu'avant mais avec un tableau direct ce qui
permet de faire des sélections discontinues
'********************************

Public Function SumCompMatR2(Val as Integer, ParamArray OtherArgs())
'donc la même chose qu'avant mais en plus puissant
```


… à faire en tant qu'exercice…

```
End Function
```

Cette section toute neuve est encore très pauvre. Son ojbectif sera de présenter comment utiliser certaines fonctions Excel dont la syntaxe V.B.A. n'est pas des plus évidentes à deviner…

Commencons par les fonctions matricielles assez souvent demandées et certainement celles dont la syntaxe V.B.A. est la moins évidente à deviner.

Considérons les premières colonnes du tableau ci-dessous:

N° Client	Secteur d'activité	N° Commande	Date Commande	Article	Nombre	Prix par pièce	Rabais%	Prix total ave
100	Alimentaire	1	03.01.2000	Compaq Presario 100	12	1650	1.5%	
123	Machines/Outils	2		IBM 500	2	2299	0.0%	
109	Assurances	3	03.01.2000	AST Intel 150	5	2690	0.0%	
104	Assurances	4	03.01.2000	AST Intel 200	3	3190	0.0%	
117	Banques	5	04.01.2000	Compaq Presario 100	13	1650	1.5%	
103	Alimentaire	6	04.01.2000	AST Intel 150	2	2690	0.0%	
104	Assurances	7	04.01.2000	AST Intel 200	2	3190	0.0%	
111	Education	8	04.01.2000	IBM 500	4	2299	0.0%	
113	Construction	9		Compaq Presario 100	4	1650	0.0%	
116	Pharmaceutique	10	04.01.2000	IBM 500	2	2299	0.0%	
110	Distribution	11	05.01.2000	AST Intel 200	6	3190	1.5%	
112	Machines/Outils	12	05.01.2000	Compaq Presario 100	6	1650	1.5%	
123	Machines/Outils	13	05.01.2000	IBM 500	6	2299	1.5%	
113	Construction	14	05.01.2000	AST Intel 150	3	2690	0.0%	
115	Distribution	15	05.01.2000	Compaq Presario 100	8	1650	1.5%	
124	Assurances	16	05.01.2000	AST Intel 200	8	3190	1.5%	
124	Assurances	17		Compaq Presario 100	11	1650	1.5%	
106	Construction	18	05.01.2000	AST Intel 200	11	3190	1.5%	
101	Construction	19	05.01.2000	Compaq Presario 100	14	1650	1.5%	

Nous ne considérerons que le nombre de lignes de ce tableau et égal à 150. Nous chercherons le nombre d'unités vendues pour tous les articles IBM 500 du secteur Education.

```
Public Sub FctMat()

    Dim strNmArticle, strSecteur,strCalcul As String
    Dim intRow As Integer
    intRow=150

    strNmArticle = "IBM 500"
    strSecteur = "Education"

strCalcul = "SumProduct((b2:b" & intRow & "=""" & strSecteur & """ )*(e2:e"
& intRow & "=""" & strNmArticle & """)*(f2:f" & intRow & "))"
    MsgBox Evaluate(strCalcul)

End Sub
```


Lors de simulations de Monte-Carlo nous avons souvent besoin de récuperer le centile de données se trouvant dans un tableau, voici par exemple une fonction financière avec un tel exemple:

```
Function ValueAtRiskMC(confidence, horizon, RiskFree, StDv, StockValue)
Dim i As Integer
'According to the Black Scholes model, the price path of stocks is defined
by
'the stochastic partial differential equation dS = (r - q -1/2sigma^2)dt +
sigma dz
'where dz is a standard Brownian motion, defined by dz = epsilon * sqrt(dt)
'where epsilon is a standard normal random variable; dS is the change in
stock price
'r is the risk-free interest rate, q is the dividend of the stock,
'sigma is the volatility of the stock.

'The model implies that dS/S follows a normal distribution with mean
'r - q -1/2sigma^2, and standard deviation sigma * epsilon * sqrt(dt))
'As such the price at time 0 < t <= T is given by
'St = S0 * exp( (r – q - ½ sigma^2) dt + sigma * epsilon * sqrt(dt))
'As we are ignoring dividends etc here so
```

```
'below line is for geometric brownian motion
Dim stockReturn(1 To 10000) As Double
    For i = 1 To 10000
        stockReturn(i) = Exp((RiskFree - 0.5 * StDv ^ 2) + StDv *
Application.NormInv(Rnd(), 0, 1)) - 1
    Next i
    ValueAtRiskMC = -(horizon) ^ 0.5 * Application.Percentile(stockReturn,
1 - confidence)
    ValueAtRiskMC = StockValue * ValueAtRiskMC
End Function

'********************************
'Créateur: Vincent ISOZ
'Dernière modification: 26.02.2014
'Nom fonction: NettoieTexte()
'Commentaires: Fonction qui enlève les accentes les plus courants à la
cellule passée en argument de la fonction
'********************************

Public Function NettoieTexte(strTheTextToClean As String)

    Dim a As String * 1
    Dim B As String * 1
    Dim strExtension As String
    Dim i, j As Integer
    Dim intNbrAccents, intLongText As Integer

    Const AccChars =
"ŠŽšžŸÀÁÂÃÄÅÇÈÉÊËÌÍÎÏÐÑÒÓÔÕÖÙÚÛÜÝàáâãäåçèéêëìíîïðñòóôõöùúûüýÿ"
    Const RegChars =
"SZszYAAAAAACEEEEIIIIDNOOOOOUUUUYaaaaaaceeeeiiiidnooooouuuuyy"

    intNbrAccents = Len(AccChars)
    intLongText = Len(strTheTextToClean)

    'petit nettoyage préalabe
    strTheTextToClean = Replace(Replace(strTheTextToClean, " ", "_"), "&",
"_et_")

    'On nettoie les accents
    For i = 1 To intNbrAccents Step 1
        a = Mid(AccChars, i, 1)
        B = Mid(RegChars, i, 1)
        strTheTextToClean = Replace(strTheTextToClean, a, B)
    Next i
    strTheTextToClean = Application.Proper(strTheTextToClean)

    For i = intLongText To 1 Step -1
        strExtension = Mid(strTheTextToClean, i, 1) & strExtension
        j = j + 1
        If Mid(strTheTextToClean, i, 1) = "." Then Exit For
    Next i
    NettoieTexte = Replace(strTheTextToClean, Right(strTheTextToClean, j),
LCase(strExtension))
End Function
```

```
'*******************************
'Créateur: Vincent ISOZ
'Dernière modification: 26.02.2014
'Nom fonction: MajusculesPhrases()
'Commentaires: Fonction qui met très basiquement toutes les lettrs sans
accents qui sont après des ponctuations en majuscules. Le code peut être
facilement amélioré pour prendre aussi en compte les accentes en début de
phrase
'*******************************

Public Function MajusculesPhrases(strTheTextToClean As String)

    On Error GoTo gestionErreurs

    Dim i As Integer

    strTheTextToClean = LCase(strTheTextToClean)
    strTheTextToClean = UCase(Left(strTheTextToClean, 1)) &
Right(strTheTextToClean, Len(strTheTextToClean) - 1)

    For i = 97 To 122 Step 1
        strTheTextToClean = Replace(strTheTextToClean, ". " & Chr(i), ". "
& UCase(Chr(i)))
        strTheTextToClean = Replace(strTheTextToClean, "? " & Chr(i), "? "
& UCase(Chr(i)))
        strTheTextToClean = Replace(strTheTextToClean, "! " & Chr(i), "! "
& UCase(Chr(i)))
        strTheTextToClean = Replace(strTheTextToClean, ": " & Chr(i), ": "
& UCase(Chr(i)))
    Next i

    MajusculesPhrases = strTheTextToClean

    Exit Function
gestionErreurs:
    MajusculesPhrases = ""

End Function
```

10. PROCÉDURES (exemples)

À nouveau, dans la même idée que pour les fonctions, un petit échantillon de procédures résumant certaines commandes importantes de V.B.A. pour Excel. Évidememnt en théorie cette section est sans fin en termes de possibilités mais il faut la voir (pour rappel) comme des notes de cours de commandes qui me sont souvent demandées.

Avant de commencer signalons l'importance de la commande (en matière d'optimisation):

```
Application.ScreenUpdating = False
```

qui permet d'éviter que l'écran affiche les étapes de la macro et permet ainsi de gagner du temps en exécution. Mais ne pas oublier de remettre à True vers ou à la fin de la procédure!!

Il en va de même pour les calculs:

```
Application.Calculation = xlCalculationManual
```

qui permet d'éviter qu'Excel rafraichisse les calculs à chaque fois alors que ce n'est nécessaire qu'à la fin. Donc ne pas oublier lorsque le code est terminé de mettre:

```
Application.Calculation = xlCalculationAutomatic
```

Avant de commencer je souhaite communiquer la table suivante:

Interior	Font	HTML	RED	GREEN	BLUE	COLOR
	[Color 0]	#FFFFFF	255	255	255	[Color 0]
	[Color 1]	#000000	0	0	0	[Color 1]
		#FFFFFF	255	255	255	[Color 2]
	[Color 3]	#FF0000	255	0	0	[Color 3]
	[Color 4]	#00FF00	0	255	0	[Color 4]
	[Color 5]	#0000FF	0	0	255	[Color 5]
	[Color 6]	#FFFF00	255	255	0	[Color 6]
	[Color 7]	#FF00FF	255	0	255	[Color 7]
	[Color 8]	#00FFFF	0	255	255	[Color 8]
	[Color 9]	#800000	128	0	0	[Color 9]
	[Color 10]	#008000	0	128	0	[Color 10]
	[Color 11]	#000080	0	0	128	[Color 11]
	[Color 12]	#808000	128	128	0	[Color 12]
	[Color 13]	#800080	128	0	128	[Color 13]
	[Color 14]	#008080	0	128	128	[Color 14]
	[Color 15]	#C0C0C0	192	192	192	[Color 15]
	[Color 16]	#808080	128	128	128	[Color 16]
	[Color 17]	#9999FF	153	153	255	[Color 17]
	[Color 18]	#993366	153	51	102	[Color 18]
	[Color 19]	#FFFFCC	255	255	204	[Color 19]
	[Color 20]	#CCFFFF	204	255	255	[Color 20]
	[Color 21]	#660066	102	0	102	[Color 21]
	[Color 22]	#FF8080	255	128	128	[Color 22]
	[Color 23]	#0066CC	0	102	204	[Color 23]
	[Color 24]	#CCCCFF	204	204	255	[Color 24]
	[Color 25]	#000080	0	0	128	[Color 25]
	[Color 26]	#FF00FF	255	0	255	[Color 26]
	[Color 27]	#FFFF00	255	255	0	[Color 27]
	[Color 28]	#00FFFF	0	255	255	[Color 28]
	[Color 29]	#800080	128	0	128	[Color 29]
	[Color 30]	#800000	128	0	0	[Color 30]
	[Color 31]	#008080	0	128	128	[Color 31]
	[Color 32]	#0000FF	0	0	255	[Color 32]
	[Color 33]	#00CCFF	0	204	255	[Color 33]
	[Color 34]	#CCFFFF	204	255	255	[Color 34]
	[Color 35]	#CCFFCC	204	255	204	[Color 35]
	[Color 36]	#FFFF99	255	255	153	[Color 36]
	[Color 37]	#99CCFF	153	204	255	[Color 37]
	[Color 38]	#FF99CC	255	153	204	[Color 38]
	[Color 39]	#CC99FF	204	153	255	[Color 39]
	[Color 40]	#FFCC99	255	204	153	[Color 40]
	[Color 41]	#3366FF	51	102	255	[Color 41]
	[Color 42]	#33CCCC	51	204	204	[Color 42]
	[Color 43]	#99CC00	153	204	0	[Color 43]
	[Color 44]	#FFCC00	255	204	0	[Color 44]
	[Color 45]	#FF9900	255	153	0	[Color 45]
	[Color 46]	#FF6600	255	102	0	[Color 46]
	[Color 47]	#666699	102	102	153	[Color 47]
	[Color 48]	#969696	150	150	150	[Color 48]
	[Color 49]	#003366	0	51	102	[Color 49]
	[Color 50]	#339966	51	153	102	[Color 50]
	[Color 51]	#003300	0	51	0	[Color 51]
	[Color 52]	#333300	51	51	0	[Color 52]
	[Color 53]	#993300	153	51	0	[Color 53]
	[Color 54]	#993366	153	51	102	[Color 54]
	[Color 55]	#333399	51	51	153	[Color 55]
	[Color 56]	#333333	51	51	51	[Color 56]

Et pour changer le couleur d'une cellule proprement:

Cells(i, 1).Interior.Color = RGB(i, i, i)

```
Option Explicit

'ci-dessous, une variable que nous utiliserons pour l'étude des modules de
classe (faites-ne abstraction pour l'instant)
Dim z_label() As New ClsZoneLabel

'********************************
'Créateur: Vincent ISOZ
'Dernière modification: 27.10.2003
'Nom procédure: ajoutsupp()
'Commentaires: Utilisation des macros automatiques pour
'le code V.B.A. (modification manuelle également)
'Objectif du cours: rappel sur quelques options disponibles
```

```
'dans MS Excel et introduction à quelques commandes utiles
'********************************
Public Sub AjoutSupp()

    'On va cacher toutes les éléments inutiles à l'écran
    With ActiveWindow
        'On cache la grille
        .DisplayGridlines = False
        'On cache les libellés de cellule
        .DisplayHeadings = False
        'On cache la barre de défilement horizontale
        .DisplayHorizontalScrollBar = False
        'Pareil avec la verticale
        .DisplayVerticalScrollBar = False
        'On cache les feuilles
        .DisplayWorkbookTabs = False
    End With
    With Application
        'On cache la barre de formules
        .DisplayFormulaBar = False
        'On cache la barre d'état
        .DisplayStatusBar = False
        'On cache les fenêtres multiples de la barre des tâches
        .ShowWindowsInTaskbar = False
        'On cache la barre d'outils
        .CommandBars("Standard").Visible = False
        'On cache la barre de formatage
        .CommandBars("Formatting").Visible = False
        'On cache l'assistant office
        .Assistant.Visible = False
        'On désactive les messages d'erreurs (nous les générerons nous-
même)
        .DisplayAlerts = False 'Désactiver les messages d'avertissement ou
d'erreurs
        'On active le calcul automatique
        .Calculation=xlAutomatic
    End With
    'On peut rajouter une commande qui permet de dire bonjour...
    MsgBox "Bonjour " & Application.UserName
    'On raffiche la barre d'état
    Application.DisplayStatusBar = True
    'On y écrit le nom de l'entreprise
    Application.StatusBar = "IT University Copyright"
    'On informe l'utilisateur sur l'état de sa machine et de son système
d'exploitation
    MsgBox "Vous êtes dans " & Application.Name & " " & Application.Version
& " " & Application.OperatingSystem
    Select Case Left(Application.Version, 2)
      Case "10"
         Msgbox "Excel 2002: Trop vieux!"
      Case "11"
         Msgbox "Excel 2003: Trop vieux!"
      Case "12"
         Msgbox "Excel 2007: Trop buggé!"
      Case "13"
         Msgbox "Excel 2010: OK!"
      Case Else
         Msgbox "Version inconnue!"
    End Select
    MsgBox "Votre imprimante par défaut est: " & Application.ActivePrinter
```

```
    'On contrôle la langue de l'application (1033=Anglais, 1036=Français,
1031=German)
    If Application.LanguageSettings.LanguageID(msoLanguageIDUI) = 1033 Then
        MsgBox "L'application est paramétrée en anglais"
    End If
    'On affiche la feuille invisible
    'On ne spécifie pas la classe car elle est triviale
    Sheets("NomFeuille").Visible = True
    'Par la méthode Range on sélectionne les cellules voulues dans la
feuille active
    Range("A1:J110").Select
    'On affiche l'outil "Grille" mis à disposition dans Excel
    Sheets("MVBA1").ShowDataForm
    'On cache la feuille avec les données
    Sheets("NomFeuille").Visible = False
    'On redirige mS Excel pour lui dire dans quelle feuille il doit revenir
    'une fois le tout effectué
    Sheets("NomFeuille").Select
    'On demande à l'utilisateur de choisir une vue (commande très très
utile!!)
    Application.Dialogs(xlDialogCustomViews).Show
    'On peut afficher un message à l'utilisateur comme quoi son fichier va
être maintenant sauvegardé
    MsgBox "Votre Document va être maintenant sauvegardé"
    'On peut mettre une ligne de code en commentaire quand on veut
effectuer des essais
    'Par exemple la ligne ci-dessous qui va sauver le classeur
    ActiveWorkbook.Save

End Sub
```



```
'*******************************
'Créateur: Vincent ISOZ
'Dernière modification: 07.07.2004
'Nom procédure: AfficheListeFeuilles()
'Commentaires: On affiche la liste des feuilles du classeur actif dans une
msgbox
'*******************************

Public Sub AfficheListeFeuilles()

    Dim shtWorksheet as Worksheet

    For each shtWorksheet In ActiveWorkbook.Worksheets
        MsgBox shtWorksheet.Name
    Next shtWorksheet

End Sub

'*******************************
'Créateur: Vincent ISOZ
'Dernière modification: 10.10.2004
'Nom procédure: AllerFeuiller()
'Commentaires: demande vers quelle feuille l'utilisateur souhaite aller
'Objectif du cours: apprendre la gestion des saisies et les boucles sur les
objets
'*******************************

Public Sub AllerFeuiller()
    Dim FeuilleXL As Worksheet, BlattName As String
```

```
    NomFeuille = InputBox("Veuillez donner le nom de la feuille!",
"Recherche de feuille")
    For Each FeuilleXL In Sheets
        'Instr permet de chercher la position d'un caractère ou mot dans
une chaîne
        If InStr(LCase(FeuilleXL .Name), LCase(NomFeuille)) > 0 Then
            FeuilleXL .Activate
            Exit Sub
        End If
    Next
    MsgBox "Aucune feuille de ce nom n'a été trouvée", vbInformation
End Sub
```

```
'********************************
'Créateur: Vincent ISOZ
'Dernière modification: 28.10.2003
'Nom procédure: factorielle()
'Commentaires: Calcule la factorielle (encore...)
'Objectif du cours: apprendre l'appel des procédures et
'la portée des variables
'********************************

Public Sub factorielle()

    Dim n, init, cible As Integer
    Dim result As Double
    Dim resp As Byte

2   n = InputBox("Valeur de n?")
    On Error GoTo 1
    init = n
    If n = 0 Then
        MsgBox "Factorielle 0!=1"
    Else
        'Appelle la fonction factrecFuc plus bas
        result = factrecFuc(n)
        MsgBox "Factorielle " & init & "!=" & result
    End If
    Range("A1").Value = result
    cible = InputBox("Choisissez le cellule ou le résultat doit être
retourné")
    Range(cible).Value = result
    Exit Sub
1   resp = MsgBox("Impossible d'exécuter la procédure", vbRetryCancel +
vbCritical)
    If resp = vbCancel Then
        Exit Sub
    ElseIf resp = vbRetry Then
        GoTo 2 'Attention cela est très dangereux (ne gère pas le conlit
des variables: y préférer le "call")
    End If

End Sub

Function factrecFuc(ByVal n As Integer)
'nous expliquerons le "byval", "byref" dans l'exercice qui suit

    If n <= 1 Then
        factrec = 1
    Else
        factrec = factrec(n - 1) * n
    End If
    End Function
```

```
'*******************************
'Créateur: Vincent ISOZ
'Dernière modification: 03.01.2005
'Nom procédure: Envoi()
'Commentaires: un grand classique pour comprendre la différence entre byRef
et byVal
'*******************************

Public Sub Envoi()

    Dim X, Y As Integer

    X = 10
    Y = 20
    Swap X, Y
    MsgBox "X Value = " & X & vbCrLf & " Y Value = " & Y

End Sub

Function Swap(ByRef X AS Integer, ByRef Y AS Integer)
    'Cela fonctionne tout aussi bien avec une sub
    'Changez le ByRef en ByVal pour voir la différence
    Dim tmp As Integer

    tmp = X
    X = Y
    Y = tmp
    'Donc en ByVal les données d'origine sont changées et on arrive en
quelque sorte à renvoyer de multiples valeurs

End Function
```


Une façon beaucoup plus élégante consiste à écrire:

```
Public Sub Envoi()

    Dim X, Y As Integer

    X = 10
    Y = 20
    Swap X, Y
    MsgBox "X Value = " & Swap(X, Y)(1) & vbCrLf & " Y Value = " & Swap(X,
Y)(2)

End Sub

Function Swap(ByVal X AS Integer, ByVal Y AS Integer) As Variant
    Dim tmp As Integer
    Dim int_Values() As Variant

    ReDim int_Values(1 To 2)

    int_Values(2) = X
    int_Values(1) = Y
    Swap = int_Values

End Function
```

```
'*******************************
'Créateur: Vincent ISOZ
'Dernière modification: 29.01.2004
'Nom procédure: AfficheFeuilles()
'Commentaires: un grand classique qu'il faut avoir fait au moins une fois….
'Le but étant d'utiliser une série d'objets et de définir leurs propriétés
'La procédure affiche toutes les feuilles cachées du classeur en cours
'*******************************

Public Sub AfficheFeuilles()

Dim sht As Worksheet

    ' Se répète à chaque élément.
    For Each sht In ActiveWorkbook.Sheets
        sht.Visible = xlSheetVisible
    Next

End Sub

'*******************************
'Créateur: Vincent ISOZ
'Dernière modification: 28.10.2003
'Nom procédure: ligndumax()
'Commentaires: on revient sur une fonction déjà connue mais ce coup-ci
'ce programme fait un petit plus
'*******************************

Public Sub LigneDuMax()

    Dim col As Integer
    Dim nblignes, valmax As Double

    col = InputBox("Tapez le numéro de la colonne")
    nblignes = Rows.Count
    valmax = Application.WorksheetFunction.Max(Columns(col))
    For i = 1 To nblignes
        If Cells(i, col) = valmax Then
            MsgBox "La ligne du contenu maximal est: " & i
            'Voir l'aide de V.B.A. pour les couleurs ou les macros
automatiques à choix
            Range(Cells(i, col), Cells(i, col)).Interior.ColorIndex = 6
            Range(Cells(i, col), Cells(i, col)).Value = "C'était ici"
            Exit For
        End If
    Next i

End Sub
```

```
'*******************************
'Créateur: Vincent ISOZ
'Dernière modification: 28.10.2003
'Nom procédure: startRND()
'Commentaires: Nous revenons sur la création de boutons sur les feuilles
'Le but étant lorsque l'on clique dessus, un nombre aléatoire est généré
dans une cellule
'*******************************

Public Sub StartRND()

    Dim nb As Integer

    nb = 0
    'on initialize le générateur de nombre aléatoires
    Randomize
    Do While nb < 2
        nb = Int(RND * 20)
    Loop
    Cells(2).Value = nb

End Sub

'*******************************
'Créateur: Vincent ISOZ
'Dernière modification: 28.10.2003
'Nom procédure: selectcase()
'Commentaires: Apprentissage de la structure conditionnelle de test "Case"
'(commande utile et importante)
'Créez un bouton pour l'exécuter
'*******************************

Public Sub SelectCase()

    Select Case Hour(Time)
        Case 0 To 6 'Ce type d'intervalles sont malheureusent borné... et
il est impossible en V.B.A. de faire des "And" dans un Case donc... pas
d'intervalles ouverts possibles!!!
            Message = "Bonne nuit..."
        Case 7
            Message = "Bonjour..."
        Case 8 To 11
            Message = "Bonne matinée..."
        Case 12, 13, 14 'La virgule fait office de "Or" et il n'y a pas de
"And" existant en V.B.A. pour le Select Case!!!
            Message = "Bon appétit..."
        Case 15 To 18
            Message = "Bon après-midi..."
        Case Is >=19 'on peut mettre de multiples "Is" séparés par des
virgules pour faire des "Or", mais encore une fois, il n'existe pas de
"And"
            Message = "Bonne soirée..."
        Case Else
            Message = "Valeur inconnue!!! Que dire..."
    End Select
    MsgBox Message

End Sub
```

```
'*******************************
'Créateur: Vincent ISOZ
'Dernière modification: 28.10.2003
'Nom procédure: Recup()
'Commentaires: cette procédure permet à nouveau d'acquérir la connaissance
'de quelques commandes importantes de V.B.A. (dont les boites de dialogue)
'*******************************

Public Sub Recup()

    Dim datedepart as Date
    Dim delaimens as Integer

    With Application
        .DisplayScrollBars = False
        .DisplayFormulaBar = False
        .Assistant.Visible = False
        .CommandBars("Standard").Visible = False
        .CommandBars("Formatting").Visible = False
        .StatusBar = "V.B.A. c'est bien:-)"
    End With
    datedepart = InputBox("Entrez la date de départ", "Mon Programme", "",
50, 50)
    delaimens = InputBox("Entrez le délai mensuel", "Mon Programme", "",
50, 50)
    'on appelle la fonction ajmois que nous avions créé dans les exercices
sur les fonctions
    resultat =ajmois(datedepart, delaimens)
    MsgBox resultat
    'Première méthode d'enregistrement (l'utilisateur ne voit rien)
    'Enregistre sous le nom du modèle de base *.xls
    ActiveWorkbook.Save
    'Enregistre sous un format spécifique avec des propriétés spécifiques
    Application.Dialogs (xlDialogSaveAs).Show

End Sub
```



```
'*******************************
'Créateur: Vincent ISOZ
'Dernière modification: 28.10.2003
'Nom procédure: Effacer()
'Commentaires: Cette procédure efface la feuille 3 du classeur et une autre
à choix et permet également d'effacer un fichier dans le dossier courant
'Objectif: Les boîtes de dialogues standards
'*******************************

Public Sub effacer()

    Dim fname As Integer

    'Désactive la demande de confirmation des suppressions
    Application.DisplayAlerts = False
On Error GoTo GestionErreurs
    Worksheets("feuil3").Delete
    fname = InputBox("Quelle feuille souhaiteriez-vous effacer?")
    Worksheets(fname).Delete
    'on crée un dossier si les droits nous le permettent
    MkDir "c:/mon_dossier"
    Application.Dialogs(xlDialogSaveAs).Show
```

```
    'il faut au préalable avoir un petit fichier préparé dans le dossier
courant
    'pour voir le fonctionnement de cette instruction
    fname = InputBox("Donnez le nom du fichier que vous souhaitez effacer",
"Suppression", "c:/")
    Kill fname
    'Evidemment, on pourrait utiliser la commande de concaténation pour
éviter à avoir c:/ qui s'affiche dans l'input box
    'Sinon une petite ligne qui permet d'effacer tous les fichiers d'un
certain type
    'Kill "disque:/dossier/sous-dossier/*.txt"
GestionErreurs:
    MsgBox Str(Err.Number) & ": " & Err.Description & Chr(13) & "Veuillez
contacter le responsable informatique", , "Erreur"

End Sub

'*******************************
'Créateur: Vincent ISOZ
'Dernière modification: 24.12.2001
'Nom procédure: NettoieLignesVides()
'Commentaires: Enlève les lignes simples ou doubles vides d'un tableau de
façon optimale
'*******************************

Sub SupprimerLignesVides()

    'On se position à la dernière ligne et on remonte
    LastRow = Cells(rows.Count, "A").End(xlUp).Row

    'On supprime les lignes en remontant plutôt qu'en descendant (c'est
plus malin!)
    For i = LastRow To 1 Step -1
        MsgBox WorksheetFunction.CountA(i)
       If WorksheetFunction.CountA(rows(i)) = 0 Then
          rows(i).Delete
       End If
    Next i

End Sub
```

```
'********************************
'Créateur: Vincent ISOZ
'Dernière modification: 24.12.2003
'Nom procédure: ajoutMenu()
'Commentaires: Ajoute un composant au menu 'Outils'
'(fonctionne dans MS Word et MS Excel)
'********************************

Public Sub ajoutMenu()

    Dim oNewLigne As CommandBarButton
    Set oNewLigne =
CommandBars("Tools").Controls.Add(Type:=msoControlButton)

    With oNewLigne
        .BeginGroup = True
        .Caption = "Fermer le document"
        .FaceId = 0
        'on appelle la procédure nommée "FemerLeDocument"... (ci-dessous)
        'on indique le nom du module dans lequel se trouve l'action
(procedure) à exécuter
        'au besoin
        .OnAction = "Procedures.FermeLeDocument"
    End With

End Sub

Sub FermeLeDocument()

    MsgBox "FERME LE DOCUMENT COURANT !!!"
    'on supprime l'élément de menu
    CommandBars("Tools").Controls("Fermer le document").Delete

End Sub

'La liste des FaceId est disponible auprès de votre formateur

'********************************
'Créateur: Vincent ISOZ
'Dernière modification: 08.07.2004
'Nom procédure: mcrOpenFile()
'Commentaires: Cette procédure récupère le nom de fichier et chemin ouvert
dans la boîte de dialogue OpenFileName
'Objectif du cours: apprendre à accéder aux fichiers
'********************************

Public Sub OuvertureFichier()
     dim fileToOpen as String
     ChDrive "C"
     ChDir "C:\" 'dossier par défaut de la commande qui suit
     fileToOpen = Application.GetOpenFilename("Text Files (*.txt), *.txt")
     If fileToOpen <> "False" Then
          MsgBox "Open " & fileToOpen
     End If

End Sub
```

```
'*******************************
'Créateur: Vincent ISOZ
'Dernière modification: 29.10.2003
'Nom procédure: FichierExiste(), Essai()
'Commentaires: procédure appelant une fonction pour vérifier si un fichier
existe
'Objectif du cours: apprendre à accéder aux fichiers
'*******************************

Public Function FichierExiste(filespec)

     ' Renvoie True si le fichier existe
     ' Ne pas oublier d'ajouter la référence Microsoft Script Runtime
     Dim fso, msg
     Set fso = CreateObject("Scripting.FileSystemObject")

     FichierExiste = IIf(fso.FileExists(filespec), True, False)

End Function

Public Sub TestFichier()

     fichier = "c:\autoexec.bat"
     If FichierExiste(fichier) Then
          MsgBox "Le fichier " & fichier " existe"
     Else
          MsgBox "Le fichier " & fichier " n'existe pas"
     End If

End Sub

'*******************************
'Créateur: Vincent ISOZ
'Dernière modification: 29.10.2003
'Nom procédure: lst_fichiers()
'Commentaires: liste dans une feuille du classeur 'les fichiers se trouvant
dans un dossier donné (très utile pour faire des listings).
'Objectif du cours: apprendre à accéder aux fichiers
'*******************************

Public Sub Lst_Fichiers()

    Dim str_Path, dept As String
    Dim i As Long

    str_Path = ActiveWorkbook.Path
    'Nous créeons une variable ainsi, au besoin nous pouvons demander
    'dans une inputbox le chemin à l'utilisateur
    str_Path = chemin & "\dossierexcel"
    'ou de manière plus élégante et élaborée
    Set str_Path = Application.FileDialog(msoFileDialogFolderPicker)
    With str_Path
        .AllowMultiSelect = False
        .Show
    End With
    str_Path = str_Path.SelectedItems(1)
    'Il faut choisir une feuille du classeur dans laquelle mettre
    'la liste des fichiers trouvés
    Sheets("liste_fichiers").Select
```

```
    'FileSearch ne fonctionne plus depuis Excel 2007. Voir le chapitre
spécifique plus bas.
    With Application.FileSearch
        'La ligne ci-dessous va permette (on ne sait jamais)
        'de réinitialiser tous les paramètres de recherche
        'd'une éventuelle recherche antérieure
        .NewSearch
        'l'endroit où doit chercher MS Excel (ou MS Word)
        .LookIn = str_Path
        'est-ce qu'il doit aller regarder aussi dans les sous-dossiers
        .SearchSubFolders = False
        'types de fichiers (même principe que l'explorateur)
        .Filename = "*.*"
        'la méthode execute va permette de controler s'il y au moins
        'un fichier de trouvé
        If .Execute() > 0 Then
            MsgBox "Il y a " & .FoundFiles.Count & " fichier(s) trouvé(s)."
            For i = 1 To .FoundFiles.Count
                'nous affichons les extensions de fichiers
                dept = Right(.FoundFiles(i), 3)
                Cells(i, "A") = .FoundFiles(i)
                Cells(i, "B") = dept
            Next i
        Else
            MsgBox "Pas de fichiers"
        End If
    End With

End Sub

'*******************************
'Créateur: Vincent ISOZ
'Dernière modification: 29.10.2003
'Nom fonction: FileOrDirExists ()
'Commentaires: Vérifier existance fichier
'*******************************

Public Function FileOrDirExists(PathName As String) As Boolean

    Dim iTemp As Integer

     'Ignore errors to allow for error evaluation
    On Error Resume Next
    iTemp = GetAttr(PathName)

     'Check if error exists and set response appropriately
    Select Case Err.Number
    Case Is = 0
        FileOrDirExists = True
    Case Else
        FileOrDirExists = False
    End Select

     'Resume error checking
    On Error GoTo 0
End Function

'*******************************
'Créateur: Vincent ISOZ
'Dernière modification: 06.05.2022
'Nom procédure: Rename_all_files_in_a_folder ()
```

```
'Commentaires: Renomme des fichiers en masse dans un dossier donné
'*******************************

Sub Renommer_fichiers_batch()

    Dim FolderNamse As String
    Dim i As Integer
    i = 1
    FolderName="c:\tmp\"

    On Error Go To ErrorHandler
    File_Name=Dir(Folder_Name)

    Do Until File_Name = ""
       Name Folder_Name & File_Name As Folder_Name & _
       File_Name & " " & i
       File_Name = Dir
       i = i + 1
    Loop
ErrorHandler:
     'Resume error checking
     Exit Sub
End Sub
```

```
'*******************************
'Créateur: Vincent ISOZ
'Dernière modification: 15.08.2005
'Nom procédure: FichierExcel()
'Commentaires: Affiche dans la feuille de calcul active, la liste des
fichiers Excel triée par ordre alphabétique du disque C
'*******************************

Public Sub FichierExcel()

    Dim i, j As Integer
    Dim TabExcel() As String

    'File search ne fonctionne plus depuis Excel 2007
    With Application.FileSearch
        .NewSearch
        .LookIn = "C:"
        .SearchSubFolders = True
        .Filename = "xls"
        .MatchTextExactly = True
        .Execute msoSortByFileName
        'on redimesionne le tableau avec deux dimensions avec autant de
lignes
        'que de fichiers trouvés
        ReDim TabExcel(.FoundFiles.Count, 1)
        For i = 1 To .FoundFiles.Count
            'on parcoure les caractères des fichiers trouvés à l'envers
            'pous séparer la partie du nom du fichier de celle contenant
            'le chemin (défini par un slash)
            For j = Len(.FoundFiles(i)) To 1 Step -1
                'si on trouve le slash
                If Mid(.FoundFiles(i), j, 1) = "\" Then
                    'on met le chemin dans la première colonne du tableau
                    TabExcel(i, 0) = Left(.FoundFiles(i), j)
                    'on met le nom du fichier dans la deuxième colonne tu
tableau
                    TabExcel(i, 1) = Right(.FoundFiles(i),
Len(.FoundFiles(i)) - j)
                    j = 1
                End If
            Next j
        Next i
        'on copie le résultat
        Range(Cells(1, 1), Cells(.FoundFiles.Count, 2)) = TabExcel
    End With

End Sub

'*******************************
'Créateur: Vincent ISOZ
'Dernière modification: 05.07.2004
'Nom procédure: CreationFichier()
'Commentaires: Crée un fichier dans un emplacement spécifique
'Objectif du cours: manipulation simple de fichiers
'*******************************

Public Sub CreationFichier()
Set NewBook = Workbooks.Add
    With NewBook
```

```
        .Title = "All Sales"
        .Subject = "Sales"
        .SaveAs Filename:="Allsales.xls"
    End With
End Sub


'********************************
'Créateur: Vincent ISOZ
'Dernière modification: 05.07.2004
'Nom procédure: CopieFichier()
'Commentaires: Copie un fichier d'un endroit vers un autre
'Objectif du cours: manipulation simple de fichiers
'********************************

Public Sub CopieFichier()

    Dim SourceFile, DestinationFile As String

    SourceFile = inputbox("Veuillez saisir le chemin du fichier a copier")
    DestinationFile = inputbox("Veuillez saisir le chemin du dossier de
destination")
    'La ligne ci-dessous effectue la copie
    FileCopy SourceFile, DestinationFile

End Sub

'********************************
'Créateur: Vincent ISOZ
'Dernière modification: 12.01.2005
'Nom procédure: EcrireFichier()
'Commentaires: ce code ecrit du texte dans un fichier txt existant ou non
'Objectif du cours: apprendre à écrire dans un fichier txt (ou xml selon
l'humeur)
'********************************

Public Sub EcrireFichier()

    Dim strPath as String

    'cette boîte de dialogue nous renvoie le nom du fichier saisi ainsi que
le chemin !
    strPath=Application.GetOpenFilename(fileFilter:="Text files (*.txt),
*.txt")
    'pour le xml cela serait fileFilter:="XML files (*.xml), *.xml"
    Open "c:\test.txt" For Output As #1
    'ne pas oublier pour le xml de commencer par <?xml version="1.0"
encoding=="iso-8859-1"?> et l'encodage en particulier pour les accents…
    Print #1, "Ceci est une ligne de texte."
    Print #1, "Ceci en est une autre" 'le retour à la ligne est
automatique!
    Close #1

End Sub
```

```
'*******************************
'Créateur: Vincent ISOZ
'Dernière modification: 08.07.2004
'Nom procédure: LireFichierTXT()
'Commentaires: ce code lit le contenu d'un fichier texte
'Objectif du cours: apprendre à lire dans un fichier txt
'*******************************

Public Sub LireFichierTXT()

     Dim fso, MyFile
     Const ForReading = 1 ' voir l'aide sur OpenAsTextStream
     Const TristateUseDefault = -2 'idem

     ' Créer un objet FileSystem
     Set fso = CreateObject("Scripting.FileSystemObject")

     ' Pointer le fichier à traiter
     Set f = fso.GetFile("f:\testfile.txt")
     ' Ouvrir le fichier en mode "ajout en fin de fichier" avec le format
par défaut
     Set MyFile = f.OpenAsTextStream(ForReading, TristateUseDefault)

     While MyFile.AtEndOfStream <> True
          s = s & MyFile.ReadLine & vbCrLf
     Wend

      MsgBox s
      ' Fermer le fichier
       MyFile.Close
       'Basiquement pour écrire dans un fichier (sans aller plus loin) on
peut aussi écrire
       'Set f = fso.CreateTextFile("c:\fichiertext.txt", True)
       'f.Writeline("Ceci est un test.")
       'f.close

End Sub
```

```
'********************************
'Créateur: Vincent ISOZ
'Dernière modification: 15.10.2005
'Nom procédure: LireFichierXML()
'Commentaires: ce code lit le contenu d'un fichier xml
'Objectif du cours: apprendre à lire dans un fichier xml
'********************************
```

Nous considérons d'abord le fichier morceau de fichier XML (test.xml) suivant à importer:

```
<?xml version="1.0"?>
      <SiteVisits>
        <Country CountryName="USA">
          <TotalVisits>1348</TotalVisits>
          <LatestVisit>1/4/2000</LatestVisit>
        </Country>
        <Country CountryName="UK">
          <TotalVisits>764</TotalVisits>
          <LatestVisit>1/4/2000</LatestVisit>
        </Country>
        <Country CountryName="Argentina">
          <TotalVisits>175</TotalVisits>
          <LatestVisit>1/2/2000</LatestVisit>
        </Country>
        <Country CountryName="Brazil">
          <TotalVisits>182</TotalVisits>
          <LatestVisit>1/4/2000</LatestVisit>
        </Country>
        <Country CountryName="Canada">
          <TotalVisits>688</TotalVisits>
          <LatestVisit>1/3/2000</LatestVisit>
        </Country>
        <Country CountryName="Denmark">
          <TotalVisits>204</TotalVisits>
          <LatestVisit>1/1/2000</LatestVisit>
```

le code correspondant utilisant DOM (Data Object Model) est:

```
Sub XMLChsrger()
    Dim oDoc As MSXML2.DOMDocument
    Dim fSuccess As Boolean
    Dim oRoot As MSXML2.IXMLDOMNode
    Dim oCountry As MSXML2.IXMLDOMNode
    Dim oAttributes As MSXML2.IXMLDOMNamedNodeMap
    Dim oCountryName As MSXML2.IXMLDOMNode
    Dim oChildren As MSXML2.IXMLDOMNodeList
    Dim oChild As MSXML2.IXMLDOMNode
    Dim intI As Integer
     Set oDoc = New MSXML2.DOMDocument
     ' Load the  XML from disk, without validating it. Wait for the load to
finish before proceeding.
    oDoc.async = False
    oDoc.validateOnParse = False
    fSuccess = oDoc.Load( ActiveWorkbook.Path & "\test.xml")

    ' Set up a row counter.
```

```
    intI = 5
     ' Delete the previous information.
    ActiveSheet.Cells(4, 1).CurrentRegion.ClearContents
    ' Create column headers.
    ActiveSheet.Cells(4, 1) = "Country"
    ActiveSheet.Cells(4, 2) = "Total Visits"
    ActiveSheet.Cells(4, 3) = "Latest Visit"
     ' Get the root of the XML tree.
    Set oRoot = oDoc.documentElement
     ' Go through all children of the root.
    For Each oCountry In oRoot.childNodes
      ' Collect the attributes for this country/region.
      Set oAttributes = oCountry.Attributes
      ' Extract the country/region name and place it on the worksheet.
      Set oCountryName = oAttributes.getNamedItem("CountryName")
      ActiveSheet.Cells(intI, 1).Value = oCountryName.Text
      ' Go through all the children of the country/region node.
      Set oChildren = oCountry.childNodes
      For Each oChild In oChildren
        ' Get information from each child node to the sheet.
        If oChild.nodeName = "TotalVisits" Then
          ActiveSheet.Cells(intI, 2) = oChild.nodeTypedValue
        End If
        If oChild.nodeName = "LatestVisit" Then
          ActiveSheet.Cells(intI, 3) = oChild.nodeTypedValue
        End If
      Next oChild
      intI = intI + 1
    Next oCountry
End Sub
```

le même code pour importer un fichier XML n'ayant qu'un seul niveau de noeud:


```
ActiveWorkbook.XMLImport "C:\test.xml", Nothing, True, Range("A1")
```

```
'*******************************
'Créateur: Vincent ISOZ
'Dernière modification: 29.10.2003
'Nom procédure: copie()
'Commentaires: passe des valeurs d'une cellule à une autre
'Objectif du cours: rappel
'*******************************

Public Sub Copie()

    '1ère méthode Copie le contenu de A1 dans B1
    Cells(2).Value = Cells(1).Value
    '2ème méthode Copie le contenu de A2 dans B2
    Range("B2").Value = Range("A2").Value
    '3ème méthode Copie le contenu de A3 dans B3
    Range("b3").Select
    ActiveCell.Value = Range("a3").Value
    'Indiquons la commande ActiveCell.Row qui est très utile pour savoir à
quelle ligne se trouve la cellule active
    '4ème méthode (assez utilisée)
    Range(Cells(4, 2), Cells(4, 2)).Value = Range(Cells(4, 1), Cells(4,
1)).Value
    '5ème méthode (la plus utilisée)
    Cells(5, 2).Value = Cells(5, 1)
```

```
End Sub

'*******************************
'Créateur: Vincent ISOZ
'Dernière modification: 29.10.2003
'Nom procédure: id(), idok(), idnul()
'Commentaires: Ce programme a pour objectif l'appel de procédure par
d'autres procédures
'La routine id() appelle la routine idok() et idnul()
'Objectif du cours: rappel
'*******************************

Public Sub id()

    reponse = InputBox("Identifiez vous:", "ID Box", "Nom Utilisateur",
100, 100)
    If reponse = "Maud" Or reponse = "maud" Or reponse = "MAUD" Then
        idok
    ElseIf reponse Like "*soz" Then
        idok
    Else
        idnul
    End If

End Sub

Public Sub idok()

    With Application
        .StandardFont = "Comic Sans MS"
        .StandardFont = 12
        .ShowToolTips = False
    End With
    ActiveWindow.WindowState = xlNormal

End Sub

Public Sub idnul()

    With Application
        .StandardFont = "Arial"
        .StandardFont = 10
        .ShowToolTips = False
    End With
    ActiveWindow.WindowState = xlMaximized

End Sub
```

```
'*******************************
'Créateur: Vincent ISOZ
'Dernière modification: 29.10.2003
'Nom procédure: disposercol()
'Commentaires: Cette procédure prend le contenu des 6 premières colonnes,
dispose
'le tout dans la colonne A et trie en ordre croissant ce qui s'y trouve
'Objectif du cours: rappel
'*******************************

Public Sub DisposerCol()

    Dim nbcolonnes, LignemaxA, LignemaxB As Integer
    Dim nblignes As Double

    'On compte le nombre de lignes de la version d'Excel
    nblignes = Rows.Count
    'On s'occupe que des colonnes A à F
    'On fait la boucle autant de fois qu'il n'y a de colonnes
    For c = 1 To nbcolonnes Step 1
        'On sélectionne les colonnes A à F
        Columns("a:f").Select
        'Ici on trie les colonnes de façon ascendante relativement aux
valeurs contenues dans la colonne A
        Selection.Sort Key1:=Range("A1"), Order1:=xlAscending,
Header:=xlGuess, _
        OrderCustom:=1, MatchCase:=False, Orientation:=xlTopToBottom
        'On cherche la valeur max de la colonne A ce qui nous donnera la
dernière ligne ayant un contenu
        valmax = Application.Max(Columns(1))
        'Maintenant qu'on sait qu'elle est la plus grande valeur de la
colonne A on va chercher à qu'elle ligne cette dernière se trouve
        For r = 1 To nblignes
            If Cells(r, 1) = valmax Then
                LignemaxA = r + 1
                Exit For
            End If
        Next r
        'Pour la prochaine boucle on remet la valeur de r à 1
        r = 1
        'On fait de même avec la colonne qui suit on la trie pour éviter
d'avoir des cellules vides
        'On écrit (c+1) car on s'occupe de la colonne qui suit...
        Columns(c + 1).Select
        'on la trie
        Selection.Sort Key1:=Cells(1, c + 1), Order1:=xlAscending,
Header:=xlGuess, _
        OrderCustom:=1, MatchCase:=False, Orientation:=xlTopToBottom
        'On regarde jusqu'à qu'elle ligne il y a des valeurs dans la
colonne qui suit
        For r = 1 To nblignes
            If Cells(r, c + 1) = Application.Max(Columns(c + 1)) Then
                LignemaxB = r
                Exit For
            End If
        Next r
        'Maintenant que la colonne suivante est prête on va la
couper/coller à la suite de la colonne A
        Range(Cells(1, c + 1), Cells(LignemaxB, c + 1)).Select
        Selection.Cut
```

```
        Cells(LignemaxA, 1).Select
        ActiveSheet.Paste
        'Comme on devra déterminer la nouvelle longeur de la colonne A on
la retre
    Next c

End Sub

'********************************
'Créateur: Vincent ISOZ
'Dernière modification: 29.10.2003
'Nom procédure: nettoyage()
'Commentaires: Programme qui nettoie les doublons et trie un tableau de
contenu numérique quelconque mais le test de comparaison se fait sur 3
colonnes uniquement
'Ensuite, un graphique est créé et s'adapte automatiquement au nombre de
cellules
'Une fois le graphique fait, on affichage la boîte de dialogue
d'enregistrement
'Object du cours: exercice récapitulatif
'********************************

Public Sub Nettoyage()

    Dim n, i, J, k As Double

    'on suppose que la feuille active est celle qui contient les trois
colonnes
    For n = 1 To Rows.Count Step 1
        If Cells(n, 1) = "" Then
            Exit For
        End If
    Next n
    For i = 2 To n - 1 Step 1
        For J = i + 1 To n - 1 Step 1
            If Cells(i, 1) = Cells(J, 1) And Cells(i, 2) = Cells(J, 2) And
Cells(i, 3) = Cells(J, 3) Then
                Application.Rows(J).Delete
            End If
        Next J
    Next i
    Range("A1").Select
    Selection.Sort Key1:=Range("A1"), Order1:=xlAscending, Header:=xlGuess,
_
            OrderCustom:=1, MatchCase:=False, Orientation:=xlTopToBottom
    For n = 1 To Rows.Count Step 1
        If Cells(n, 1) = "" Then
            Exit For
        End If
    Next n
    Charts.Add
    ActiveChart.ChartType = xlLineMarkers
    ActiveChart.SeriesCollection.NewSeries
    'attention selon la langue de l'interface de MS Excel de remplacer
Feuil par le nom adéquat
    plagevaleursx = "=Feuil1!R2C1:R" & n & "C1"
    'Que l'on pourrait très bien remplacer par un tableau en écrivant:
    'plagevaleursx = Array("jan", "fev",
"mars","avril","mai","juin","juillet")
    ActiveChart.SeriesCollection(1).XValues = plagevaleursx
```

```
    'attention selon la langue de l'interface de MS Excel remplacer Feuil
par le nom adéquat
    plagevaleurs = "=Feuil1!R2C3:R" & n & "C3"
    'Que l'on pourrait très bien aussi remplacer par un tableau en
écrivant:
    'plagevaleurs = Array("43","20","50","30","60","27","70")
    ActiveChart.SeriesCollection(1).Values = plagevaleurs
    'Attention à la manière d'écrire cette ligne (sinon cela rajoute une
série vide!)
    ActiveChart.SeriesCollection(1).Name = "=""Valeurs"""
    'il se peut parfois qu'on doive supprimer une série parasite
    'ActiveChart.SeriesCollection(1).Delete
     ActiveChart.Location Where:=xlLocationAsNewSheet
     'ActiveChart.SetSourceData Source:=Range(NomFeuille.Cells(1,5),
     NomFeuille.Cells(5, 6))
           :=xlColumns
    With ActiveChart
        .HasTitle = True
        .ChartTitle.Characters.Text = "Représentation Graphique"
        .Axes(xlCategory, xlPrimary).HasTitle = False
        .Axes(xlValue, xlPrimary).HasTitle = True
        .Axes(xlValue, xlPrimary).AxisTitle.Characters.Text = "Age
[années]"
    End With
    With ActiveChart.Axes(xlCategory)
        .HasMajorGridlines = False
        .HasMinorGridlines = False
    End With
    With ActiveChart.Axes(xlValue)
        .HasMajorGridlines = False
        .HasMinorGridlines = False
    End With
    ActiveChart.HasLegend = False
    ActiveChart.ApplyDataLabels Type:=xlDataLabelsShowValue,
    LegendKey:=False
    ActiveChart.PlotArea.Select
    With Selection.Border
        .ColorIndex = 16
        .Weight = xlThin
        .LineStyle = xlContinuous
    End With
    Selection.Fill.OneColorGradient Style:=msoGradientVertical, Variant:=1,
    Degree:=0.231372549019608
    With Selection
        .Fill.Visible = True
        .Fill.ForeColor.SchemeColor = 15
    End With
    ActiveChart.SeriesCollection(1).Select
    With Selection.Border
        .Weight = xlThin
        .LineStyle = xlAutomatic
    End With
    With Selection
        .MarkerBackgroundColorIndex = 2
        .MarkerForegroundColorIndex = xlAutomatic
        .MarkerStyle = xlDiamond
        .Smooth = False
        .MarkerSize = 5
        .Shadow = False
    End With
    ActiveChart.ChartTitle.Select
```

```
    With Selection.Border
        .Weight = xlHairline
        .LineStyle = xlAutomatic
    End With
    Selection.Shadow = True
    Selection.Interior.ColorIndex = xlAutomatic
    Application.Dialogs(xlDialogPrint).Show

    'Ici on va controller la taille et la position du graphique avec des
commandes spéciales sinon il fait seulement en relatif. Un pixel est en
standard égal à 1/72 de pouces.

    With ActiveChart.Parent
        .Height = 300
        .Width = 600
        .Top = 100
        .Left = 100
    End With

    'On va renommer le graphique (pour voir le nom d'un graphique Excel, on
clique sur une cellule à l'extérieur et ensuite avec la touché Ctrl on
clique sur le graphique)

        ActiveChart.Parent.Name="Mon Chart"
        'On exporte le graphique en tant qu'image (très pratique pour le
web ou InDesign)
        ActiveChart.Export Filename:="c:\Graph.gif", FilterName:="GIF"

End Sub
```

Indiquons le cas très intéressant et très demandé dans les grandes multinationales du graphique qui se positionne sur la cellule active en cours (et dont évidemment les données s'adaptent). Cela se fera sur l'énvement On_Change de la feuille:


```
Private Sub Worksheet_SelectionChange(ByVal Target As Range)
    ActiveSheet.Shapes("Nom_Graph").Left = ActiveSheet.Cells(Target.Row +
1, Target.Column + 1).Left
    ActiveSheet.Shapes("Nom_Graph ").Top = ActiveSheet.Cells(Target.Row +
1, Target.Column + 1).Top
End Sub
```

```
'*******************************
'Créateur: Vincent ISOZ
'Dernière modification: 29.10.2003
'Nom procédure: feuilles()
'Commentaires: 'Ce programme va chercher dans une feuille excel nomée
"noms"
' une liste de noms et créer autant de feuille qu'il y a de nom et chacune
de
'ses feuilles sera nommée selon le nom correspondant
'Objectif: Apprendre à manipuler des feuilles
'*******************************

Public Sub Feuilles()

    For i = 1 To Rows.Count
        If Worksheets("noms").Cells(i, 1).Value = "" Then
            Exit Sub
        End If
        Sheets.Add
        nomfeuille = Worksheets("noms").Cells(i, 1)
        'quand on insère un nom, la feuille en a toujours déjà un par
défaut
        nomdefautfeuille = "Feuil" & i
        Sheets(nomdefautfeuille).Name = nomfeuille
        'la feuille n'est jamais insérée au bon endroit
        Sheets(nomfeuille).Select
        Sheets(nomfeuille).Move After:=Sheets(1 + i)
    Next i
    'ou pour insérer une feuille en dernière position (!)
    ActiveWorkbook.Sheets.Add After:=Sheets(Sheets.Count)
    'et pour changer le nom de cette nouvelle feuille
    Sheets(Sheets.Count).Name = "Nouvelle feuille"
End Sub

'*******************************
'Créateur: Vincent ISOZ
'Dernière modification: 21.10.2005
'Nom procédure: MonteCarlo()
'Commentaires: Exemple d'utilisation de Monte-Carlo (outil très puissant)
'Objectif: faire un peu de vraie programmation
'*******************************

Public Sub MonteCarlo()
    Dim n, p As Integer
    Randomize
    n = 0
    p = 1000
    'on désactive la grille de la feuille active pour dessinger un
graphique
    ActiveWindow.DisplayGridlines = False
    With ActiveSheet
        'on épure la feuille de tous les objets existants
        .DrawingObjects.Delete
        'on dessine un rectangle d'une taille de 300x300 et avec un trait
d'épaisseur 10
        .Shapes.AddShape msoShapeRectangle, 10, 10, 300, 300
        'on dessine un arce ce cercle de rayon 300
        .Shapes.AddShape msoShapeArc, 10, 10, 300, 300
        For i = 1 To p
```

```
            'on crée les points avec la function Random de 0 à 1 dans
l'écart 0 à 300
            'du rectangle y compris la largeur des bords de celui-ci
             x = Int(Rnd * 300) + 10
             y = Int(Rnd * 300) + 10
             .Shapes.AddShape msoShape4pointStar, x - 2, y - 2, 5, 5
             If Sqr((x - 10) * (x - 10) + (y - 10) * (y - 10)) < 300 Then n
= n + 1
        Next i
        Cells(1, 7) = 4 * n / p
    End With
End Sub


'********************************
'Créateur: Vincent ISOZ
'Dernière modification: 29.10.2003
'Nom procédure: impfiligrane()
'Commentaires: Gadget...
'Objectif: passer le temps
'********************************

Public Sub ImpFiligrane()

    'On déclare une variable de type sélection
    Dim ZoneImpr As Range
    'On définit dès lors comme zone d'impression la zone sélectionnée
    Set ZoneImpr = Range(ActiveSheetl.PageSetup.PrintArea)

    'On fait une acquisition d'image de la zone
    ZoneImpr.CopyPicture xlScreen, xlBitmap
    'On colle cette image exactement à la place de la sélection à imprimer
    ActiveSheet.Paste Destination:=ZoneImpr
    'On regarde l'aperçu avant impression
    ActiveWindow.SelectedSheets.PrintPreview
    'On supprime l'image
    ActiveSheet.Shapes(ActiveSheet.Shapes.Count).Delete

End Sub

'********************************
'Créateur: Derk
'Dernière modification: 17.10.2012
'Nom procédure: ImportImageOnRange()
'Commentaires: Importe une image/picture dans une cellule spécifiée pour
une certaine hauteur et largeur modulable.
'********************************

Sub ImportPicture()
    With ActiveSheet.Pictures.Insert _
        ("C:\Program Files\Common Files\Microsoft
Shared\Clipart\cagcat50\AN01124_.wmf")
        .Top = Range("B4").Top
        .Left = Range("B4").Left
        .Height = Range("B10").Top - Range("B4").Top
        .Width = Range("H4").Left - Range("B4").Left
    End With
End Sub
```

```
'*******************************
'Créateur: Vincent ISOZ
'Dernière modification: 03.02.2005
'Nom procédure: OuvreFicherExcel()
'Commentaires: Cette procédure ouvre un fichier MS Excel externe dans le
but d'y écrire (ou lire) des valeurs et le masque pour pas que
l'utilisateur puisse le voir. C'est une méthode possible parmi tant
d'autres.
'*******************************

Public Sub OuvreFichierExcel1()

    Dim strPath, strFileName As string
    Dim objExpWorkb As Workbook

    strPath="C:\"
    strFileName="FichierExcelOuEcrire.xls"

    Set objExpWorkb = Excel.Workbooks.Open(strPath & strFileName)

    objExpWorkb.Worksheets("sheet1").Range("A1").Value="ça marche ! O_o"
    'ou autre possibilité utilisant le numéro de la feuille car le nom ce
n'est pas l'idéal s'il vient à changer
    objExpWorkb.Sheets(1).Range("A1").Value="ça marche ! O_o"
    'ou mieux encore car n'utilisant pas le caption de la feuille ni le
numéro car il pourrait lui aussi changer! Nous utilisons le nom V.B.A. de
la feuille
objExpWorkb.Worksheets(CStr(objExpWorkb.VBProject.VBComponents("sht_No
mVBAFeuille").Properties("Name"))).Range("A1").Value="ça marche ! O_o"

    objExpWorkb.Activate
    ActiveWindow.visible=False
    objExpWorkb.Save
    objExpWorkb.Close
    Set objExpWorkb=Nothing

End Sub

'Autre manière plus rapide car n'ouvre pas Excel à l'écran
Public Sub OuvreFichierExcel2()

    Dim strPath, strFileName As string
    Dim objExpWorkb As Workbook
    Dim objExcel As Object

    strPath="C:\"
    strFileName="FichierExcelOuEcrire.xls"

    Set objExcel = CreateObject("Excel.Application")
    Set objExpWorkb = objExcel.Workbooks.Open(strPath & strFileName)

    objExpWorkb.Worksheets("sheet1").Range("A1").Value="ça marche ! O_o"
    'ou autre possibilité utilisant le numéro de la feuille car le nom ce
n'est pas l'idéal s'il vient à changer
    objExpWorkb.Sheets(1).Range("A1").Value="ça marche ! O_o"
    'ou mieux encore car n'utilisant pas le caption de la feuille ni le
numéro car il pourrait lui aussi changer! Nous utilisons le nom V.B.A. de
la feuille
objExpWorkb.Worksheets(CStr(objExpWorkb.VBProject.VBComponents("sht_No
mVBAFeuille").Properties("Name"))).Range("A1").Value="ça marche ! O_o"
```

```
    objExpWorkb.Save
    objExpWorkb.Close
    Set objExcel=Nothing
    Set objExpWorkb=Nothing

End Sub

'Troisième manière plus directe encore
Public Sub OuvreFichierExcel2()

    Dim strPath, strFileName As String
    Dim objExpWorkb As Workbook
    Dim objExcel As Object

    strPath = "C:\Users\Isoz Vincent\Desktop\"
    strFileName = "FichierExcelOuEcrire.xlsx"

    Set objExcel = GetObject(strPath & strFileName)

    objExcel.Sheets(1).Range("A1").Value = "ça marche ! O_o"

    objExcel.Save
    objExcel.Close
    Set objExcel = Nothing
    Set objExcel = Nothing

End Sub

'********************************
'Créateur: Vincent ISOZ
'Dernière modification: 29.10.2003
'Nom procédure: ouvreword()
'Commentaires: Cette procédure copie un graphique dans MS Word et y écrit
un petit texte
'Objectif: montrer comment lancer une instance d'un autre logiciel
'********************************

Public Sub OuvreWord()

    'il faut avoir au préalable installé la référence Microsoft Word 9.0
Object Library
    'Pour cela dans VBAE aller dans le menu outils/références

    Set appliword = CreateObject("Word.application")

    Sheets("Feuil1").Select
    ActiveSheet.ChartObjects(1).Select
    'pour sélectionner une zone de cellules non vides automatiques on
utilisera la commande:
    'ActiveSheet.UsedRange.Select
    ActiveChart.ChartArea.Copy
    'On pourrait aussi écrire:
    'Worksheets("Sheet1").Range("A1:D4").Copy
    With appliword
        .Documents.Add DocumentType:=wdNewBlankDocument
        .Visible = True
        .Selection.TypeText Text:="Hello World"
        .Selection.Paste
        .activedocument.Save
        .Quit
```

```
    End With

End Sub

'*******************************
'Créateur: Vincent ISOZ
'Dernière modification: 10.01.2006
'Nom procédure: CreerRDVOutlook()
'Commentaires: Cette procédure montre comment ouvrir Outlook et y créer un
rendez-vous
'*******************************

Public Sub CreerRDVOutlook()

    Dim olApp As Object 'Référence à outlook
    Dim olAppointment As Object 'Référence aux rdv outlook
    Const olAppointmentItem = 1

    'Création du lien vers outlook
    Set olApp = CreateObject("Outlook.Application")
    Set olAppointment = olApp.CreateItem(olAppointmentItem)

    'Détails du rdv
    With olAppointment
        .Subject = "Discuter du contrat"
        .Start = DateSerial(2006, 2, 24) + TimeSerial(9, 30, 0)
        .End = DateSerial(2006, 2, 24) + TimeSerial(11, 30, 0)
        .ReminderPlaySound = True
        .Save
    End With
    'Quitte Outlook
    olApp.Quit
    'Vide la mémoire de l'objet
    Set olApp = Nothing

End Sub

'*******************************
'Créateur: Vincent ISOZ
'Dernière modification: 08.07.2004
'Nom procédure: ExShell()
'Commentaires: Comment exécuter n'importe quoi  (programme, batch,…) en
ligne de commande
'*******************************

Public Sub ExShell()

    'Cette procédure permet de lancer Word, écrire dans sa zone de travail
et ensuite de quitte Word

    'A adapter en fonction de votre installation
    app = "C:\Program Files\Microsoft Office\Office\WINWORD.EXE"
    'Exécute Word ; 3 = "en plein écran"
    ret = Shell(app, 3)
    'Pour ouvrir la calcultrice Windows: Shell("calc.exe",1)
    SendKeys "bonjour Word!", True
    SendKeys "%fq", True ' Réalise ALT+F+Q ; True=reste dans Word
    'SendKeys "%{F4}N", True' Réalise ALT+F4 et répond Non pour
    'quitter sans enregistrer

End Sub
```

```
'*******************************
'Créateur: Vincent ISOZ
'Dernière modification: 16.07.2004
'Nom procédure: ExeAutomat()
'Commentaires: procédure qui s'exécute automatique a chaque intervalle de
temps donné
'*******************************

Public Sub ExeAutomat()

      Application.OnTime Now + TimeValue("00:00:15"), "hello"

End Sub

Public Sub hello()

     MsgBox "Hello World"
     ExeAutomat

End Sub
```

Remarque:

A ce niveau du cours votre formateur va vous rappeler certaines notions entre VB 6.0, VB.Net, V.B.A. et VBScript.

Code pratique en VBScript (cours d'initiation de 2 jours) pour au démarrage de l'ordinateur (au fait l'idée est plutôt d'utiliser les tâches automatiques de Windows pour lancer le script à une heure donnée sinon il suffit de mettre le fichier dans le dossier *Démarrage*…), ouvrir un fichier MS Excel et exécuter une macro spécifique à l'intérieur de celui-ci:


```
Set Xl = CreateObject("Excel.application")
Xl.Visible = True
Set Wb = Xl.Workbooks.Open("C:\test.xls")
Xl.Run "test"
'Xl.ActiveWindow.SelectedSheets.PrintOut pour imprimer
'WScript.Sleep (60*1000)
'Xl.Application.Save pour sauvegarder
```

Si on veut faire un script qui se lance toutes les minutes:

```
Do
   WScript.Sleep (60*1000)

   Set Xl = CreateObject("Excel.application")
   Xl.Visible = False
   Set Wb = Xl.Workbooks.Open("C:\test.xls")
   Xl.Run "test"
Loop
```

```
'********************************
'Créateur: http://spreadsheetpage.com/
'Dernière modification: 26.02.2014
'Nom fonction: MajusculesPhrases()
'Commentaires: Routine V.B.A. qui permet de générer un résumé des mots
(dictionnaire des termes) avec une statistique d'apparition pour chacun
'********************************
Sub MakeWordList()
    Dim InputSheet As Worksheet
    Dim WordListSheet As Worksheet
    Dim PuncChars As Variant, x As Variant
    Dim i As Long, r As Long
    Dim txt As String
    Dim wordCnt As Long
    Dim AllWords As Range
    Dim PC As PivotCache
    Dim PT As PivotTable

    MsgBox "Toutes vos textes doivent être par défaut dans la colonne A",
vbCritical + vbOKOnly
    Application.ScreenUpdating = False
    Set InputSheet = ActiveSheet
    Set WordListSheet = Worksheets.Add(after:=Worksheets(Sheets.Count))
    'If you change the "All Words" also change it later in the pivot table
V.B.A. code
    WordListSheet.Range("A1") = "All Words"
    WordListSheet.Range("A1").Font.Bold = True
    InputSheet.Activate
    wordCnt = 2
    PuncChars = Array(".", ",", ";", ":", "'", "!", "#", _
        "$", "%", "&", "(", ")", " - ", "_", "--", "+", _
        "=", "~", "/", "\", "{", "}", "[", "]", """", "?", "*")
    r = 1

'   Loop until blank cell is encountered
    Do While Cells(r, 1) <> ""
'       covert to UPPERCASE
        txt = UCase(Cells(r, 1))
'       Remove punctuation
        For i = 0 To UBound(PuncChars)
            txt = Replace(txt, PuncChars(i), "")
        Next i
'       Remove excess spaces
        txt = WorksheetFunction.Trim(txt)
'       Extract the words
        x = Split(txt)
        For i = 0 To UBound(x)
            WordListSheet.Cells(wordCnt, 1) = x(i)
            wordCnt = wordCnt + 1
        Next i
    r = r + 1
    Loop

'   Create pivot table
    WordListSheet.Activate
    Set AllWords = Range("A1").CurrentRegion
    Set PC = ActiveWorkbook.PivotCaches.Add _
        (SourceType:=xlDatabase, _
        SourceData:=AllWords)
    Set PT = PC.CreatePivotTable _
```

```
        (TableDestination:=Range("C1"), _
        TableName:="PivotTable1")
    With PT
        .AddDataField .PivotFields("All Words")
        .PivotFields("All Words").Orientation = xlRowField
    End With
    ActiveSheet.Name = "StatWords"
End Sub
```

Pour arrêter le script dans le gestionnaire des tâches il suffit d'arrêter le process *wscript*

```
'*******************************
'Créateur: https://www.extendoffice.com/documents/excel/5182-excel-list-all-table-names.html
'Dernière modification: 03.05.2018
'Nom fonction: ListTables()
'Commentaires: Routine V.B.A. qui permet de lister toutes les tables d'un classeur Excel
'*******************************
Sub ListTables()
    Dim xTable As ListObject
    Dim xSheet As Worksheet
    Dim I As Long
    I = -1
    Sheets.Add.Name = "Table Name"
    For Each xSheet In Worksheets
        For Each xTable In xSheet.ListObjects
            I = I + 1
            Sheets("Table Name").Range("A1").Offset(I).Value = xTable.Name
        Next xTable
    Next
End Sub
```


10.1 Contrôler les propriétés (métadonnées) du fichier

L'exemple est un peu brut de coffre mais cela donne évidemment le nom de la commande (ce qui est le but principal!):

```
Sub PageCount()
   MsgBox ThisWorkbook.BuiltinDocumentProperties("Last Save Time")
   MsgBox ThisWorkbook.BuiltinDocumentProperties("Author")
   MsgBox ThisWorkbook.BuiltinDocumentProperties("Company")
End Sub
```

10.2 Demander où stocker un fichier (code pour tous les app MS Office)

Petite routine et fonction qui permet à l'utilisateur de choisir un dossier pour un usage ultérieur quelconque (enregistrement, ouverture, suppression, etc.). Ce code fonctionne pour toutes les applications Microsoft Office!!!:

```
Sub TestDirectory( )
     MsgBox GetDirectory
End Sub

Public Function GetDirectory(Optional OpenAt As Variant) As Variant
     'Function purpose:  To Browser for a user selected folder.
     'If the "OpenAt" path is provided, open the browser at that directory
     'NOTE:  If invalid, it will open at the Desktop level
```

```
    Dim ShellApp As Object

     'Create a file browser window at the default folder
    Set ShellApp = CreateObject("Shell.Application"). _
    BrowseForFolder(0, "Please choose a folder", 0, OpenAt)

     'Set the folder to that selected.  (On error in case cancelled)
    On Error Resume Next
    GetDirectory = ShellApp.self.Path
    On Error GoTo 0

     'Destroy the Shell Application
    Set ShellApp = Nothing

     'Check for invalid or non-entries and send to the Invalid error
handler if found
     'Valid selections can begin L: (where L is a letter) or
     '\\ (as in \\servername\sharename.  All others are invalid
    Select Case Mid(GetDirectory, 2, 1)
    Case Is = ":"
        If Left(GetDirectory, 1) = ":" Then GoTo Invalid
    Case Is = "\"
        If Not Left(GetDirectory, 1) = "\" Then GoTo Invalid
    Case Else
        GoTo Invalid
    End Select

    Exit Function

Invalid:
     'If it was determined that the selection was invalid, set to False
    GetDirectory = False

End Function
```


10.3 Compter le nombre de page imprimées

Petite routine utile pour compter le nombre de pages avant impression:

```
Sub PageCount()
   MsgBox
(ActiveSheet.HPageBreaks.Count+1)*(ActiveSheet.VPageBreaks.Count+1) & "
pages  seront imprimées."
End Sub
```

10.4 Désactiver le clic droit de la souris

Pour désactiver le clic droit de la souris définitivement dans MS Excel (même après redémarrage de l'ordinateur) il suffit d'utiliser cette routine au moment de votre choix (ouverture du fichier par exemple). Cette routine à de plus l'avantage de désactiver le raccourci clavier qui affiche le menu contextuel:

```
Sub DisableAllShortcutMenus()
   Dim cb As CommandBar
   For Each cb In CommandBars
      If cb.Type = msoBarTypePopup Then
          cb.Enabled = False
      End If
```

```
    Next cb
End Sub
```

Ou sur l'évenement de feuille suivant (cas le plus utilisé) mais qui n'empêche pas l'utilisation du raccourci clavier qui ouvre le menu contextuel:

```
Private Sub Worksheet_BeforeRightClick (ByVal Target As Excel.Range, Cancel
As Boolean)
    Cancel = True
    MsgBox "The shortcut menu is not available."
End Sub
```

10.5 Contrôles sur feuilles

Dans la plupart des logiciels MS Office une barre d'outils nommée *Boîte d'outils contrôles* est disponible. Beaucoup des éléments disponibles peuvent être personnalisés et gérés simplement de manière très similaire aux userforms (accès à la valeur par la propriété .value de certains contrôles comme pour les scrollbars, etc.).

Cependant le plus demandé reste (au même titre que pour les userfoms) la liste déroulante. Nous proposons ici un exemple d'un code qui crée automatiquement une liste déroulante lorsque l'utiliser fait un double clic sur la colonne A (et pas ailleurs!) d'une feuille choisie à l'avance.

Lorsque le choix est fait dans la liste déroulante, celle-ci affichera une valeur dans la cellule aussi adjacente et la liste déroulante disparaît.

Voici d'abord le code à ajouter dans la feuille incriminée (…):


```
'on attrape le double clic à la volée
Private Sub Worksheet_BeforeDoubleClick(ByVal Target As Range, Cancel As
Boolean)

    'on vérifie si le double clique se fait sur la colonne A
    If Not Intersect(Target, Columns("A")) Is Nothing Then
        'on apelle une procédure se trouvant dans le module 1 en lui
passant le Range de la cellule activée
        Call Module1.AddDropDown(Target)
        Cancel = True
    End If

End Sub
```

Dans le Module1 nous trouvons:

```
Public Sub AddDropDown(Target As Range)

    Dim ddBox As DropDown
    Dim vaProducts As Variant
    Dim i As Integer

    vaProducts = Array("VBA", "VB.Net", "VB 6", "VBScript")
    'On créé la liste déroulante
    With Target
        'Attention à la feuille cible spécifiée ici !
        Set ddBox = Sheet2.DropDowns.Add(.Left, .Top, .Width, .Height)
    End With
```

```
    'on appelle une procédure et on remplit la liste déroulante lorsquelle
est activée
    With ddBox
        'on appelle la procédure
        .OnAction = "EnterProdInfo"
        'on remplit la liste déroulante
        For i = LBound(vaProducts) To UBound(vaProducts)
            .AddItem vaProducts(i)
        Next i
    End With
End Sub

Private Sub EnterProdInfo()

    Dim vaPrices As Variant
    'on déclare un tableau avec un contenu qui sera écrit juste à droit de
la cellule
    'activée
    vaPrices = Array(2400, 3800, 2600, 3400)

    With Sheet2.DropDowns(Application.Caller)
        'la difficulté dans cet exemple se situe particulièrement sur
l'utilisation
        'de la commande TopLeftCell (pas évident à connaître)
        'on écrit dans la cellule où se trouve la liste déroulante
        'la valeur .List sélectionnée dans la liste grâce à .ListIndex
        .TopLeftCell.Value = .List(.ListIndex)
        'on écrit dans la cellule adjacente le prix du cours correspondant
        .TopLeftCell.Offset(0, 1).Value = vaPrices(.ListIndex - 1)
        'on supprime la liste déroulante de la cellule où elle se trouve
        .Delete
    End With
End Sub
```


10.6 Suppression des apostrophes (travailler avec des sélections)

Lors de l'export de SAP ou d'autres progiciels, les apostrophes parasites peuvent apparaître. Malheusement la fonction Rechercher/Remplacer ne marche pas pour ce typse de situation. Il faut alors utiliser la procédure suivante mise à disposition par Microsoft:

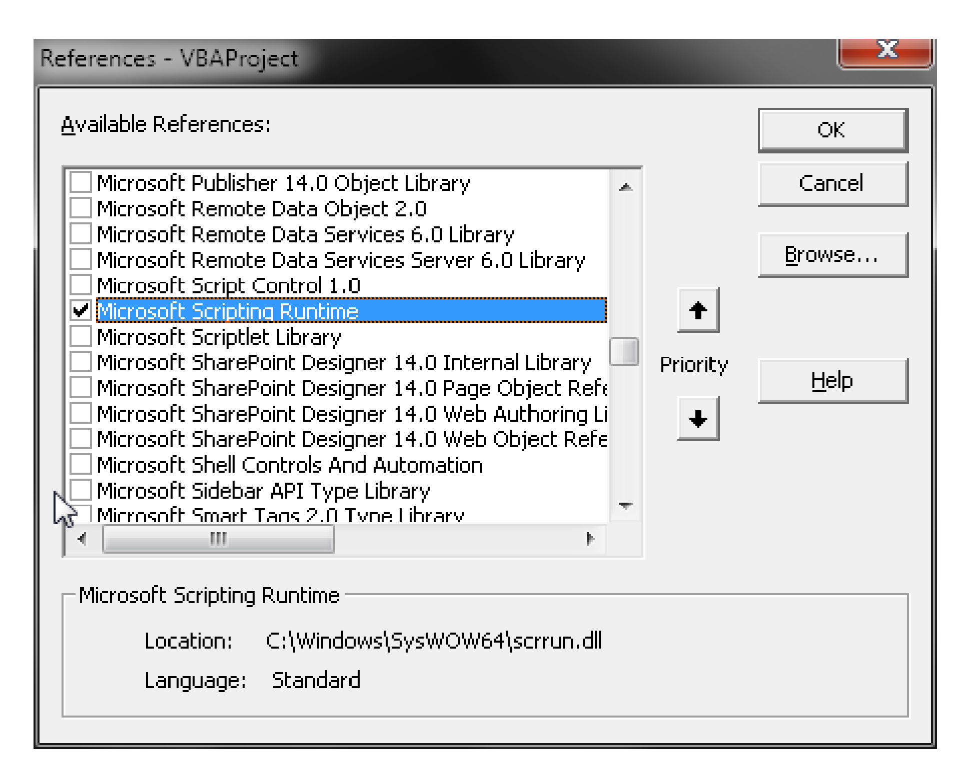
```
Public Sub RemoveApostrophe()
      'Cas important qui montre comment travailler avec une sélection!!!
       For Each CurrentCell In Selection
           If CurrentCell.HasFormula = False Then
               'Checks to make sure that procedure does not
               'change cell with a formula to be only the value
               CurrentCell.Formula = CurrentCell.Value
           End If
       Next

   End Sub
```

10.7 Tableaux associatifs

Souvent dans le cadre de l'utilisation d'un fichier Excel (ou Microsoft Office Project) plutôt que de boucler non-stop sur un tableau dont la structure ne change pas (ou peu) il est plus intéressant de stocker les données dans une structurée dite "tableau associatif" qui permet dans certaines situations d'accéder de manière très rapide à des informations précises.

Pour utiliser les tableaux associatifs en VBA, vous devrez activer la référence Microsoft Scripting runtime:



Ensuite un exemple simple de dictionnaire sera:


```
Sub TableauAssociatifDemo()
    Dim oD As New Dictionary
    Dim element
    'Les dictionnaires ne peuvent stocker que des paires d'éléments!
    'le deuxième élément pouvant être un string, un single, un array ou
autre
    oD.Add 4, "Tâche 1"
    oD.Add 7, "Tâche 3"

    'on parcoure tout pour montrer que l'on peut parcourir
    For Each element In oD
        MsgBox element & ":" & oD(element)
    Next

    'Aller chercher une info donnée
    MsgBox oD(7)

    'Test si une info existe
    MsgBox oD.Exists(8)

End Sub
```

10.8 Tableaux croisés dynamiques (TCD/PVT)

10.8.1 Filtrage des TCD

Commençons par le petit code permettant de filtrer de manière statistique, c'est-à-dire qu'il est nécessaire de lister tous les attributs dans le code pour que le filtrage fonctionne:

```
Public Sub FilterFixedPVT()

    On Error Resume Next

    With ActiveSheet.PivotTables("tcd_Ventes").PivotFields("Article")

        .PivotItems("AST Intel 150").Visible = True
        .PivotItems("AST Intel 200").Visible = False
        .PivotItems("Compaq Presario 100").Visible = False
        .PivotItems("IBM 500").Visible = False

    End With
End Sub
```

Ou de façon dynamique ne nécessitant pas de lister tous les attributs:

```
Public Sub FilterDynamicPVT()
    Dim pviPVT As PivotItem

    On Error Resume Next

    With ActiveSheet.PivotTables("tcd_Ventes").PivotFields("Article")
        strItemChoix = InputBox("Merci de choisir")
        For Each pviPVT In .PivotItems
            If UCase(pviPVT.Value) = UCase(strItemChoix) Then
                pviPVT.Visible = True
            Else
                pviPVT.Visible = False
            End If
        Next pviPVT
    End With
End Sub
```


10.8.2 Mise à jour automatique des TCD (timer)

Demande très fréquente des départements qui ont des écrans plats dans les couloirs devant montrer en temps réel les indicateurs et dont les TCD sont basés non pas sur des BDD mais sur des listes/tables MS Excel (rappelons que dans ce cas l'option de mise à jour périodique par défaut du tableur est grisée!).

Voici le code à utiliser et à adapter selon vos besoins qui peut être en réalité utilisé pour tout élément nécessitant une action répétitive:

```
Sub ScheduleAnything()
    WaitHours = 0
    WaitMin = 0
    WaitSec = 10
    NameOfThisProcedure = "ScheduleAnything" 'Recursive
    NameOfScheduledProc = "RefreshTCD"
    NextTime = Time + TimeSerial(WaitHours, WaitMin, WaitSec)
    Application.OnTime EarliestTime:=NextTime,
Procedure:=NameOfThisProcedure
    Application.Run NameOfScheduledProc
End Sub

Sub RefreshTCD()
    Sheets("PVT").PivotTables("PivotTable1").PivotCache.Refresh
End Sub
```

10.8.3 Nettoyage des TCD

Nous avons vu dans le cours TCD (tableau croisé dynamique) que lorsque nous supprimions les données du Datamart d'un TCD, celles-ci restaient stockées dans le cache de MS Excel.

Pour nettoyer ce cache il faut utiliser la routine suivante pour en nettoyer un:

```
Public Sub DeletePVTItems()
    Dim pt As PivotTable
    Set pt = ActiveSheet.PivotTables.Item(1)
    pt.PivotCache.MissingItemsLimit = xlMissingItemsNone
    'and after refresh the pivot table
End Sub
```

Ou pour tous les nettoyer:

```
Public Sub DeleteMissingItemsExcel()
    Dim pt As PivotTable
    Dim ws As Worksheet
    Dim pc As PivotCache

    'change the settings
    For Each ws In ActiveWorkbook.Worksheets
      For Each pt In ws.PivotTables
        pt.PivotCache.MissingItemsLimit = xlMissingItemsNone
      Next pt
    Next ws

    'refresh all the pivot caches
    For Each pc In ActiveWorkbook.PivotCaches
      On Error Resume Next
      pc.Refresh
    Next pc
End Sub
```


10.8.4 Protection avancée des TCD

De nombreuses options avancées de protection des TCD (au fait quasiment toutes) sont disponibles seulement en V.B.A. En voici un exemple très important:

```
Sub RestrictPivotTable()
Dim pf As PivotField
With ActiveSheet.PivotTables(1)
    .EnableWizard = False
    .EnableDrilldown = True
    .EnableFieldList = True
    .EnableFieldDialog = False
    .PivotCache.EnableRefresh = False
    For Each pf In .PageFields
        With pf
            .DragToPage = False
            .DragToRow = False
            .DragToColumn = False
            .DragToData = False
            .DragToHide = False
        End With
    Next pf
End With
```

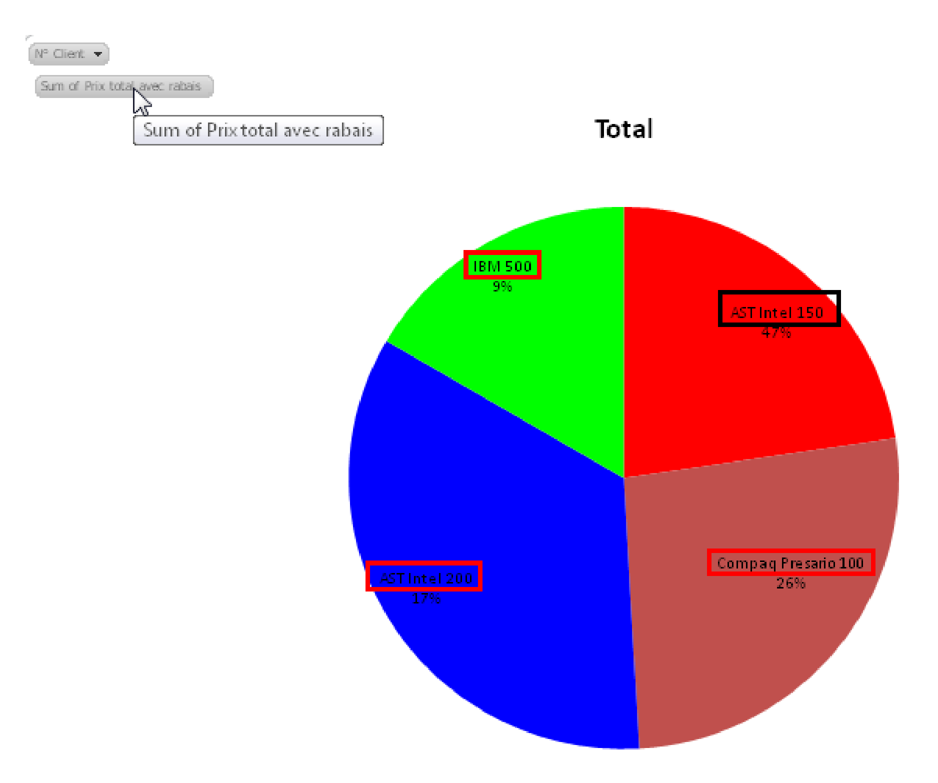
```
End Sub
```

Il est assez fou que ces options soient pour la majorité utilisable qu'à partir du V.B.A. C'est fortement dommage!

10.8.5 Préservation de la couleur des GCD

Un problème majeur des graphiques croisés dynamiques (à ce jour du moins...) c'est de perdre les couleurs d'origine dès qu'un filtrage est effectué sur le graphique croisé dynamique (ou sur le tableau croisé dynamique lié).

Comme le langage V.B.A. n'arrive pas à changer les couleurs des secteurs en s'attaquant directement aux numéros d'identifiants des tranches il faut alors passer par une astuce qui consiste à s'attaquer indirectement aux tranches en passant par les labels. Alors activezd'abord ceux-ci:



Une fois ceci fait, dans l'événement de la feuille correspondante du graphique croisé dynamique, écrivez le code suivant:

```
Sub Chart_Calculate()
'Copyright 1999 MrExcel.com
'This macro will re-color the pie slices in a chart
'So that slices for a specific category are similarly colored
'You must activate the chart labels first!
'
' Find the number of pie slices in this chart
    NumPoints = ActiveChart.SeriesCollection(1).Points.Count
    ' Loop through each pie slice
    For x = 1 To NumPoints
        ThisPt = ActiveChart.SeriesCollection(1).Points(x).DataLabel.Text
        ' Based on the label of this slice, set the color
        Select Case ThisPt
            Case "AST Intel 150"
```

```
                ActiveChart.SeriesCollection(1). _
                    Points(x).Interior.ColorIndex = 3
            Case "IBM 500"
                ActiveChart.SeriesCollection(1). _
                    Points(x).Interior.ColorIndex = 4
            Case "AST Intel 200"
                ActiveChart.SeriesCollection(1). _
                    Points(x).Interior.ColorIndex = 5
            Case "Compaq Presario 100"
                ActiveChart.SeriesCollection(1). _
                    Points(x).Interior.ColorIndex = 6
            Case Else
                ' Add code here to handle an unexpected label
        End Select
    Next x
End Sub
```

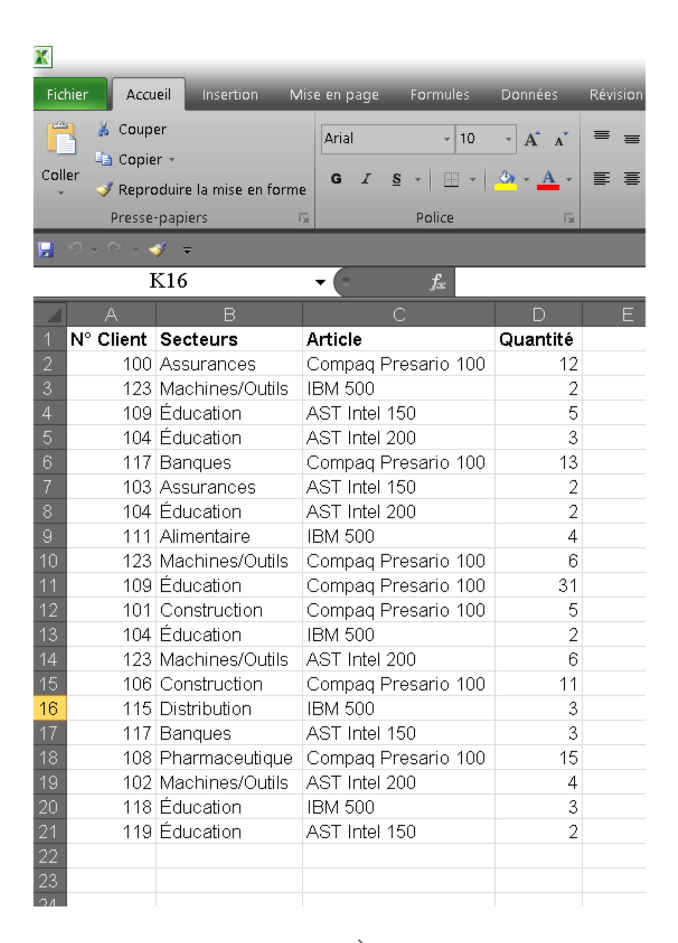
10.8.6 Filtres de slicers sur cubes TCD différents

Il s'agit du cas le plus complexe. D'abord mettons en place le scénario!

Nous avons deux feuilles avec deux jeux de données:

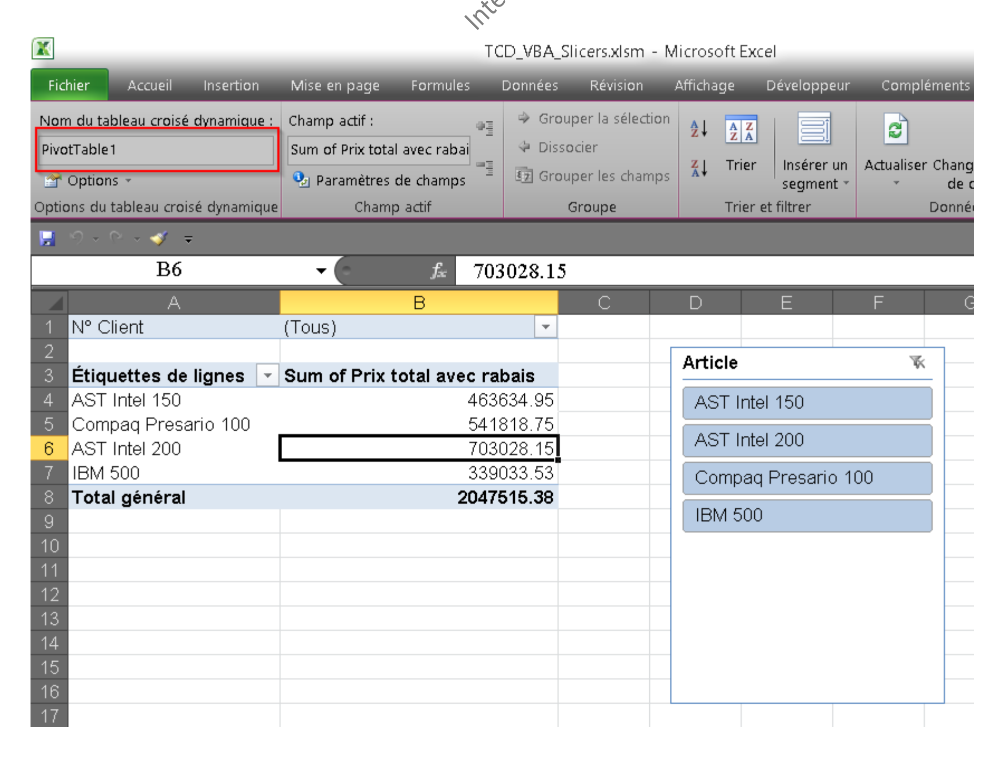
N° Client	Secteurs	N° de Commande	Date de commande	Article	Quantité	Prix par pièce	Rabais%	Prix total avec rabais	Facture payée
100	Assurances	1	2000-07-13	Compaq Presario 100	12	1650	1.50%	19503	Oui
123	Machines/Outils	2	2004-08-13	IBM 500	2	2299	0.00%	4598	Oui
109	Éducation	3	2002-08-07	AST Intel 150	5	2690	0.00%	13450	Oui
104	Éducation	4	2003-08-29	AST Intel 200	3	3190	0.00%	9570	Oui
117	Banques	5	2003-01-31	Compaq Presario 100	13	1650	1.50%	21128.25	Oui
103	Assurances	6	2001-02-09	AST Intel 150	2	2690	0.00%	5380	Oui
104	Éducation	7	2001-01-07	AST Intel 200	2	3190	0.00%	6380	Oui
111	Alimentaire	8	2004-06-05	IBM 500	4	2299	0.00%	9196	Oui
113	Construction	9	2001-12-26	Compaq Presario 100	4	1650	0.00%	6600	Oui
116	Pharmaceutique	10	2001-11-04	IBM 500	2	2299	0.00%	4598	Oui
110	Distribution	11	2003-09-23	AST Intel 200	6	3190	1.50%	18852.9	Oui
112	Machines/Outils	12	2002-02-09	Compaq Presario 100	6	1650	1.50%	9751.5	Oui
123	Machines/Outils	13	2004-12-27	IBM 500	6	2299	1.50%	13587.09	Oui
113	Construction	14	2003-07-17	AST Intel 150	3	2690	0.00%	8070	Oui
115	Distribution	15	2003-01-21	Compaq Presario 100	8	1650	1.50%	13002	Oui
124	Éducation	16	2001-10-20	AST Intel 200	8	3190	1.50%	25137.2	Oui
124	Éducation	17	2001-03-07	Compaq Presario 100	11	1650	1.50%	17877.75	Oui
106	Construction	18	2001-06-06	AST Intel 200	11	3190	1.50%	34563.65	Oui
101	Construction	19	2004-08-24	Compaq Presario 100	14	1650	1.50%	22753.5	Non
116	Pharmaceutique	20	2000-05-30	IBM 500	7	2299	1.50%	15851.605	Non
112	Machines/Outils	21	2001-06-23	AST Intel 150	6	2690	1.50%	15897.9	Oui
125	Construction	22	2002-12-29	Compaq Presario 100	23	1650	3.00%	36811.5	Oui
100	Assurances	23	2003-04-13	IBM 500	3	2299	0.00%	6897	Oui
125	Construction	24	2001-09-21	AST Intel 200	2	3190	0.00%	6380	Oui
104	Éducation	25	2002-06-29	AST Intel 150	12	2690	1.50%	31795.8	Oui
126	Machines/Outils	26	2000-04-15	AST Intel 150	24	2690	3.00%	62623.2	Oui
121	Pharmaceutique	27	2002-05-07	IBM 500	8	2299	1.50%	18116.12	Non
114	Distribution	28	2004-09-23	AST Intel 200	9	3190	1.50%	28279.35	Oui
103	Assurances	29	2004-12-10	Compaq Presario 100	6	1650	1.50%	9751.5	Oui
125	Construction	30	2001-08-09	AST Intel 200	4	3190	0.00%	12760	Oui
120	Banques	31	2000-03-30	AST Intel 150	2	2690	0.00%	5380	Oui
111	Alimentaire	32	2004-07-31	Compaq Presario 100	16	1650	3.00%	25608	Oui

Et:



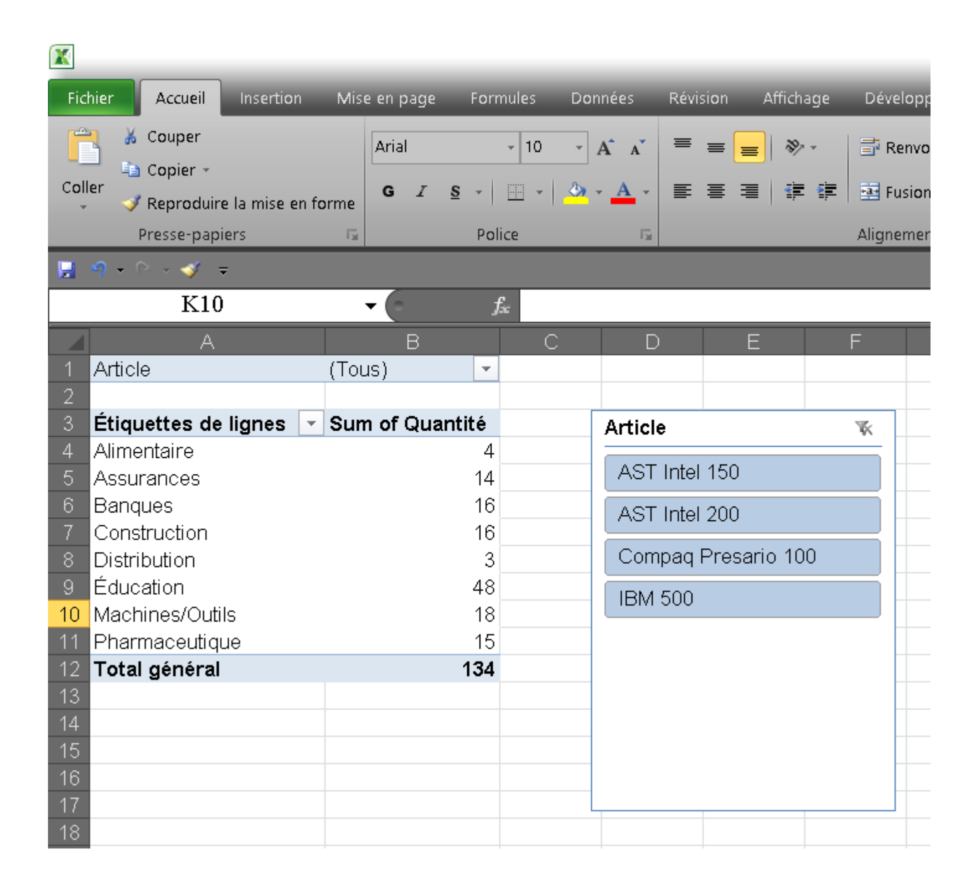
N° Client	Secteurs	Article	Quantité
100	Assurances	Compaq Presario 100	12
123	Machines/Outils	IBM 500	2
109	Éducation	AST Intel 150	5
104	Éducation	AST Intel 200	3
117	Banques	Compaq Presario 100	13
103	Assurances	AST Intel 150	2
104	Éducation	AST Intel 200	2
111	Alimentaire	IBM 500	4
123	Machines/Outils	Compaq Presario 100	6
109	Éducation	Compaq Presario 100	31
101	Construction	Compaq Presario 100	5
104	Éducation	IBM 500	2
123	Machines/Outils	AST Intel 200	6
106	Construction	Compaq Presario 100	11
115	Distribution	IBM 500	3
117	Banques	AST Intel 150	3
108	Pharmaceutique	Compaq Presario 100	15
102	Machines/Outils	AST Intel 200	4
118	Éducation	IBM 500	3
119	Éducation	AST Intel 150	2

Avec les deux tableaux croisés dynamiques respectifs sur deux feuilles différentes:

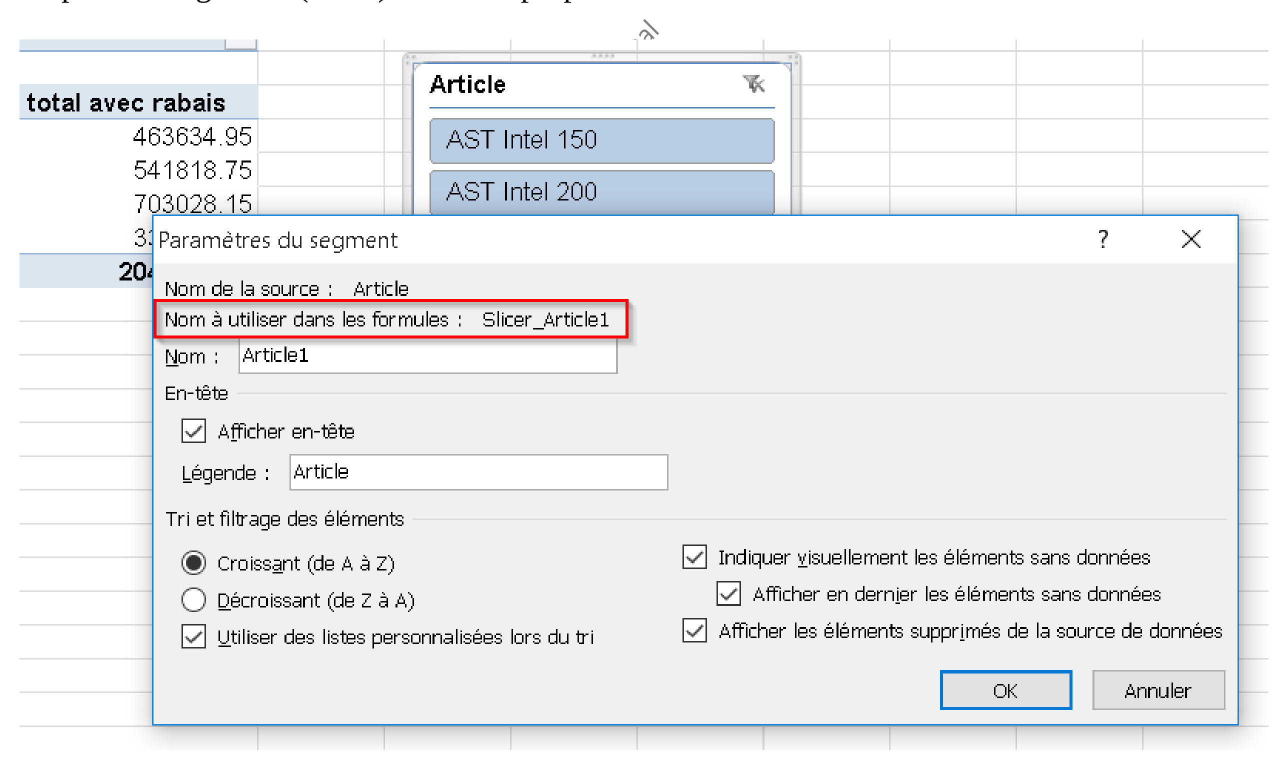


N° Client	(Tous)
Étiquettes de lignes	**Sum of Prix total avec rabais**
AST Intel 150	463634.95
Compaq Presario 100	541818.75
AST Intel 200	703028.15
IBM 500	339033.53
Total général	**2047515.38**

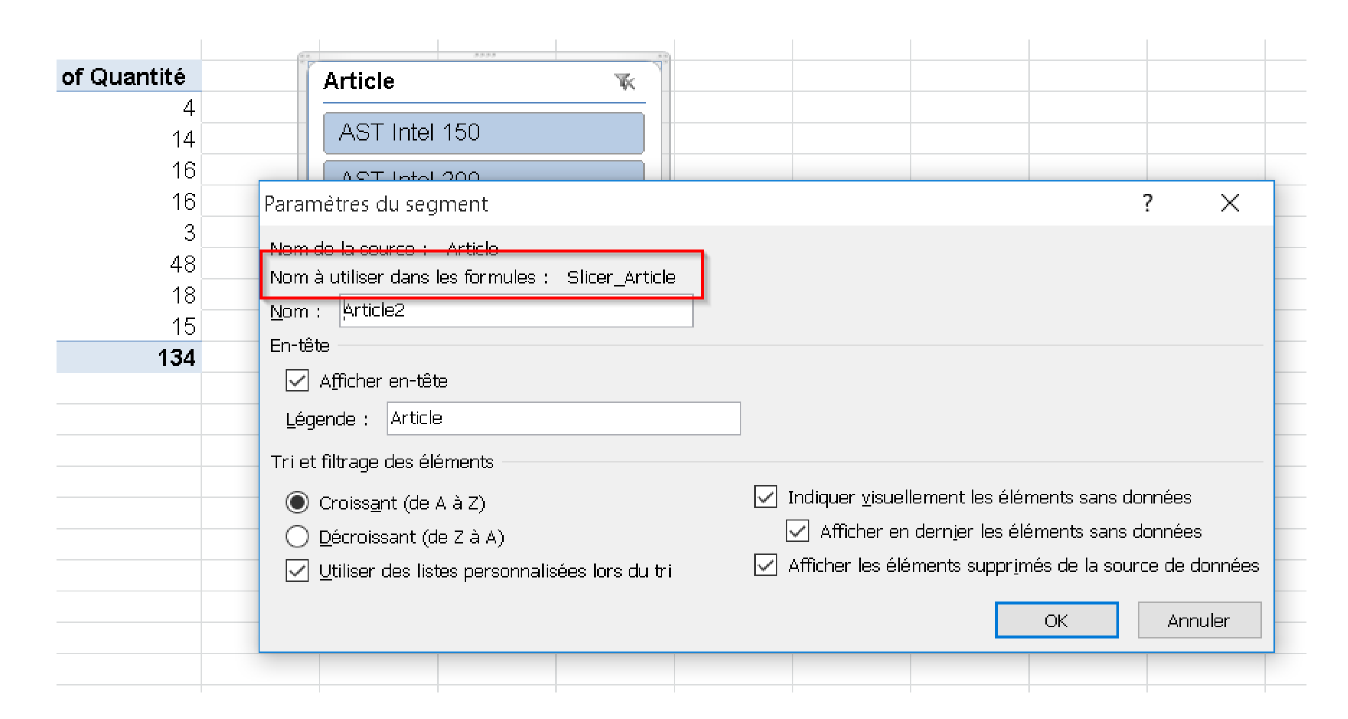
Et:



Le premier segement (slicer) a comme propriétés:

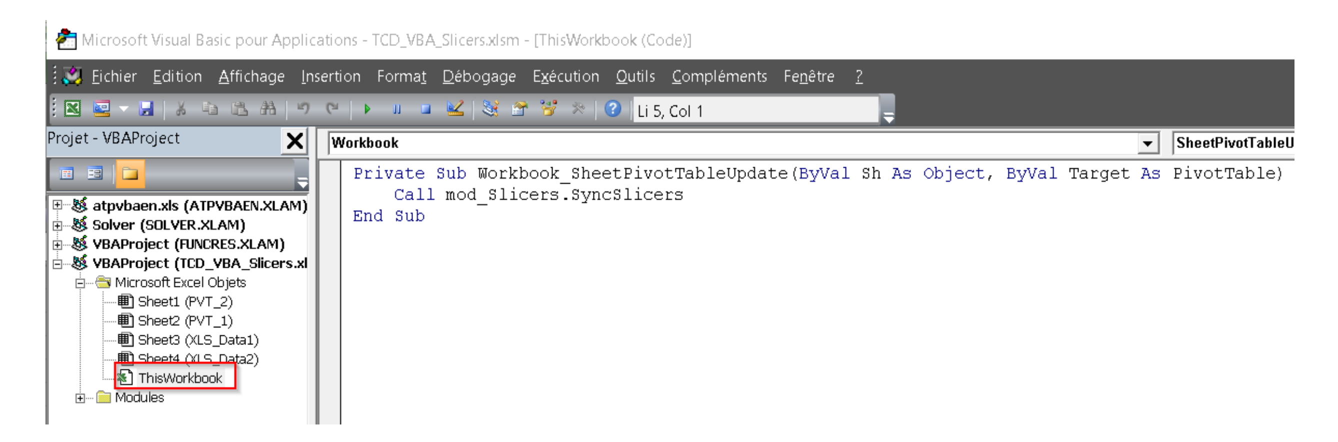


Et le deuxième:



Comme nous le savons, nous ne pouvons pas connecter deux segments (slicers) dont le jeu de données (le cube OLAP) diffère. Dès lors nous allons devoir faire usage du V.B.A.

Maintenant dans l'object **ThisWorkbook** mettez l'événement suivant:



Et dans un module de votre choix mettez le code suivant (attention à changer le nom des segments et tableaux croisés dynamiques en conséquence!):

```
Public Sub SyncSlicers()
    If ActiveSheet.Name = "PVT_1" Then
        Call Sync_Slicers( _
        slSource:=ActiveWorkbook.SlicerCaches("Slicer_Article1"), _
        slDest:=ActiveWorkbook.SlicerCaches("Slicer_Article"))
    Else
        Call Sync_Slicers( _
        slSource:=ActiveWorkbook.SlicerCaches("Slicer_Article"), _
        slDest:=ActiveWorkbook.SlicerCaches("Slicer_Article1"))
    End If
End Sub

Private Function Sync_Slicers(slSource As SlicerCache, slDest As
SlicerCache) As Boolean
```

```
'--syncs the slicer items in slDest to the Selected state of the item in
slSource
'--returns True if sucessful, False if not able to sync

    Dim sli As SlicerItem
    Dim bFound As Boolean, bNotOrphan As Boolean

    '--ensure will result in at least one item selected in slDest
    '--check if visible item in slSource is found in slDest
    On Error Resume Next
    For Each sli In slSource.VisibleSlicerItems
        bFound = Not (IsError(slDest.SlicerItems(sli.Caption)))
        If bFound Then
            With slDest.SlicerItems(sli.Caption)
                If Not .Selected Then .Selected = True
            End With
            Exit For
        End If
    Next sli
    '--if not, check if any already visible items in slDest don't exist in
slSource
    If Not bFound Then
        For Each sli In slDest.VisibleSlicerItems
            bNotOrphan = Not (IsError(slSource.SlicerItems(sli.Caption)))
            If Not bNotOrphan Then
                bFound = True
                Exit For
            End If
        Next sli
    End If
    '--if not alert and exit.
    If Not bFound Then
        MsgBox "Syncing slicers would leave no items selected in
SlicerCache: " _
            & slDest.Name, vbExclamation, "Slicer sync aborted"
    Else
        For Each sli In slSource.SlicerItems
            With slDest.SlicerItems(sli.Caption)
                If .Selected <> sli.Selected Then .Selected = sli.Selected
            End With
        Next
    End If
    On Error GoTo 0
    Sync_Slicers = bFound
End Function
```


10.9 Power Query

Le problème le plus fréquent avec Power Query est de supprimer les requêtes. Il vaut mieux ne pas le faire avec une boucle car certaines requêtes dépendent d'autres et alors il faut supprimer dans un ordre bien précis!

```
    ActiveWorkbook.Queries("Nom de la requête").Delete
```

10.10 Audit des changements sur fichier

Il est fréquent que des utilisateurs de MS Excel veuillent un code V.B.A. qui liste les changements faits sur un fichier avec le nom de l'utilisateur, le lieu de la modification et le type de modification.

Dans un module on copie le code suivant après avoir créé une feuille nommée **Audit**:

```
Option Explicit

Public Sub Auditing(ByVal strCells As String, ByVal strOldValue As String,
ByVal strNewValue As String)

    Dim strFeuilleAvant As String

    Sheets("Audit").Cells(2, 2) = Application.UserName
    Sheets("Audit").Cells(2, 3) = Now()
    Sheets("Audit").Cells(2, 4) = Now()
    Sheets("Audit").Cells(2, 5) = ActiveSheet.Name
    If strCells <> "" Then
        Sheets("Audit").Cells(2, 6) = strCells
    End If
    If Sheets("Audit").Cells(2, 6) = "" Then
        Sheets("Audit").Cells(2, 7) = strOldValue
    End If
    If strNewValue <> "" Then
        Sheets("Audit").Cells(2, 8) = strNewValue
        Sheets("Audit").Cells(2, 1) = Sheets("Audit").Cells(2 + 1, 1) + 1
        strFeuilleAvant = ActiveSheet.Name
        Sheets("Audit").Activate
        Rows("2:2").Select
        Selection.Insert Shift:=xlUp
       'ou utiliser dans certaines situations
ActiveCell.Offset(1).EntireRow.Insert à la place
        Range("A1").Select
        Sheets(strFeuilleAvant).Activate
    End If
End Sub
```


et dans chacun des feuilles on y collera le code suivant:

```
Option Explicit

Private Sub Worksheet_Change(ByVal Target As Range)
    Dim strCells As String

    strCells = Split(Columns(Target.Column).Address(ColumnAbsolute:=False),
":")(1) & Target.Row
    Module1.auditing strCells, "", Target
End Sub

Private Sub Worksheet_SelectionChange(ByVal Target As Range)
    On Error Resume Next
    Module1.auditing "", Target, ""
End Sub
```

10.11 Add-in

Le but de ce chapitre est de voir quelques manipulations relatives aux add-in de MS Excel.

Avant tout, voici le code à utiliser pour vérifier si un certain add-in est installé (et accessoirement qu'il provient du bon endroit!) ou pas à l'ouverture du fichier:

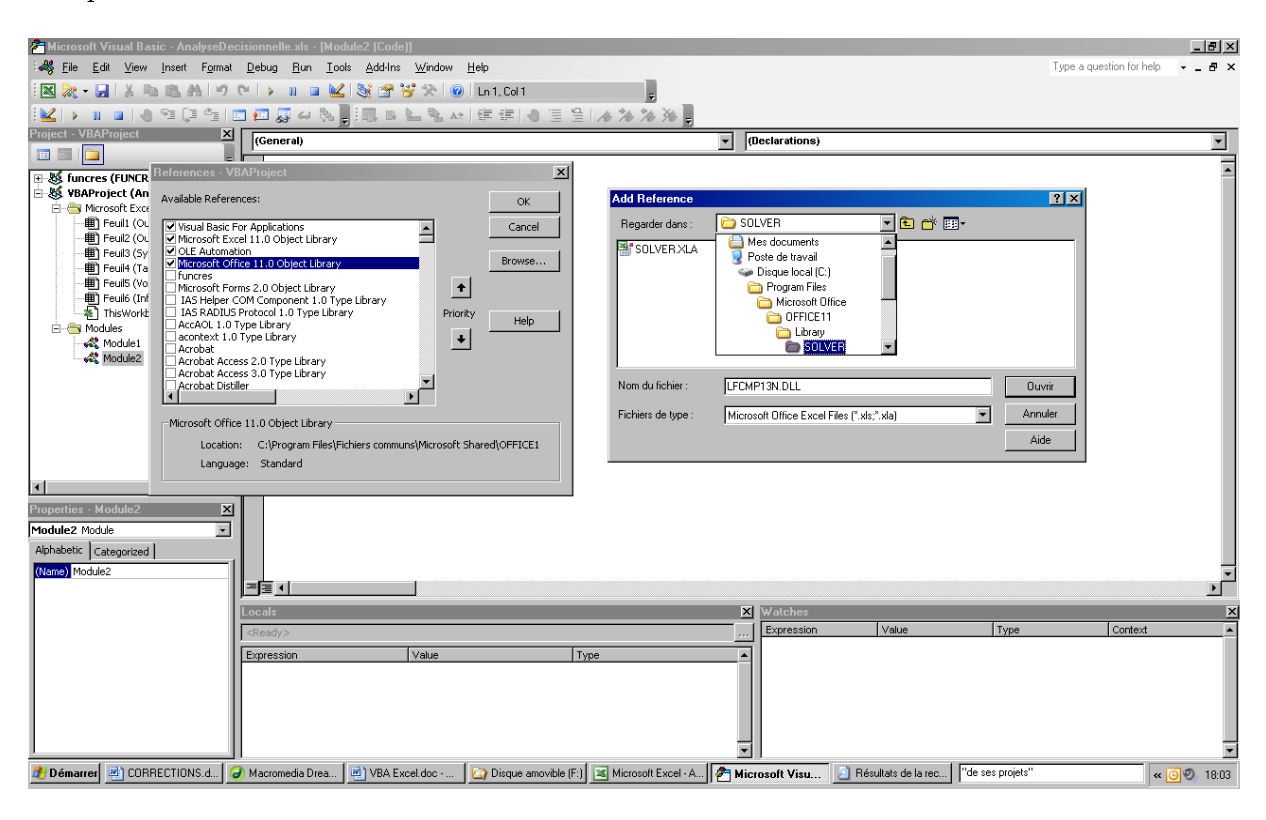
```
Private Sub Workbook_Open()
   Dim add As AddIn

   Set add=AddIns("Nom de l'addin")
   If add.Installed=True Then
       MsgBox "L'add-in est installé dans le lieu suivant: " & add.Path
   Else
       MsgBox "Add-in non installé"
   End If
End Sub
```

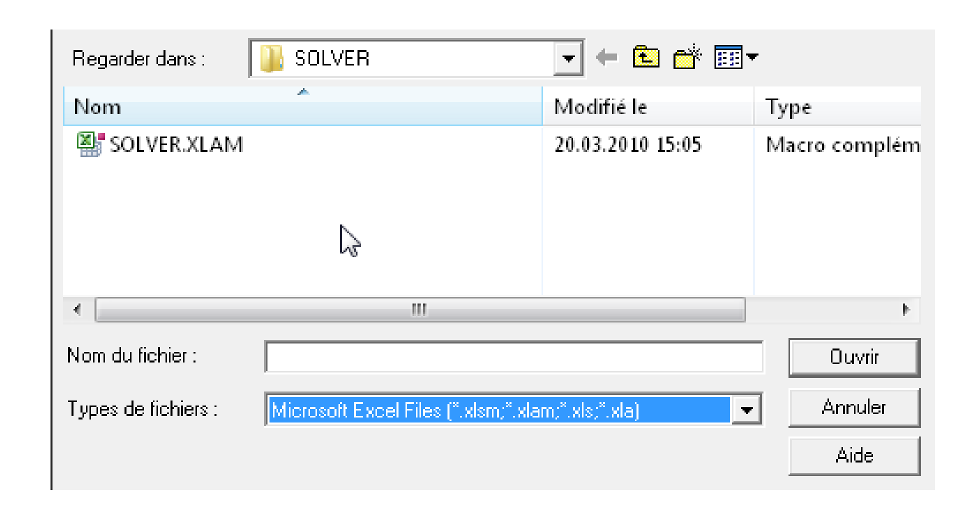
10.11.1 Appel du solveur

L'enregistrement d'une macro avec le solveur fonctionne **mais pas son exécution**!!!

Cela vient du fait qu'il faut installer la référence SOLVER.XLS dans l'éditeur VBAE comme indiqué ci-dessous:



et:



Une fois la référence ajoutée au fichier *SOLVER.XLAM*, votre code macro pourra être exécutée sans erreur. Enfin, terminons en précisant le code qui permet de faire en sorte que le solveur s'exécute sans afficher la fenêtre d'enregistrement des rapports à fin:

```
Public Sub mcrSolveur()

    SolverOk SetCell:="$F$20", MaxMinVal:=1, ValueOf:="0",
ByChange:="$B$16:$E$16"
    SolverSolve UserFinish:=True         'la partie rouge est ce qu'il faut
rajouter!
    SolverFinish KeepFinal:=1  'pour que la fenêtre de validaiton
n'apparaisse pas (car elle demande si on veut garder ou rejeter les
solutions trouvées)

End Sub
```

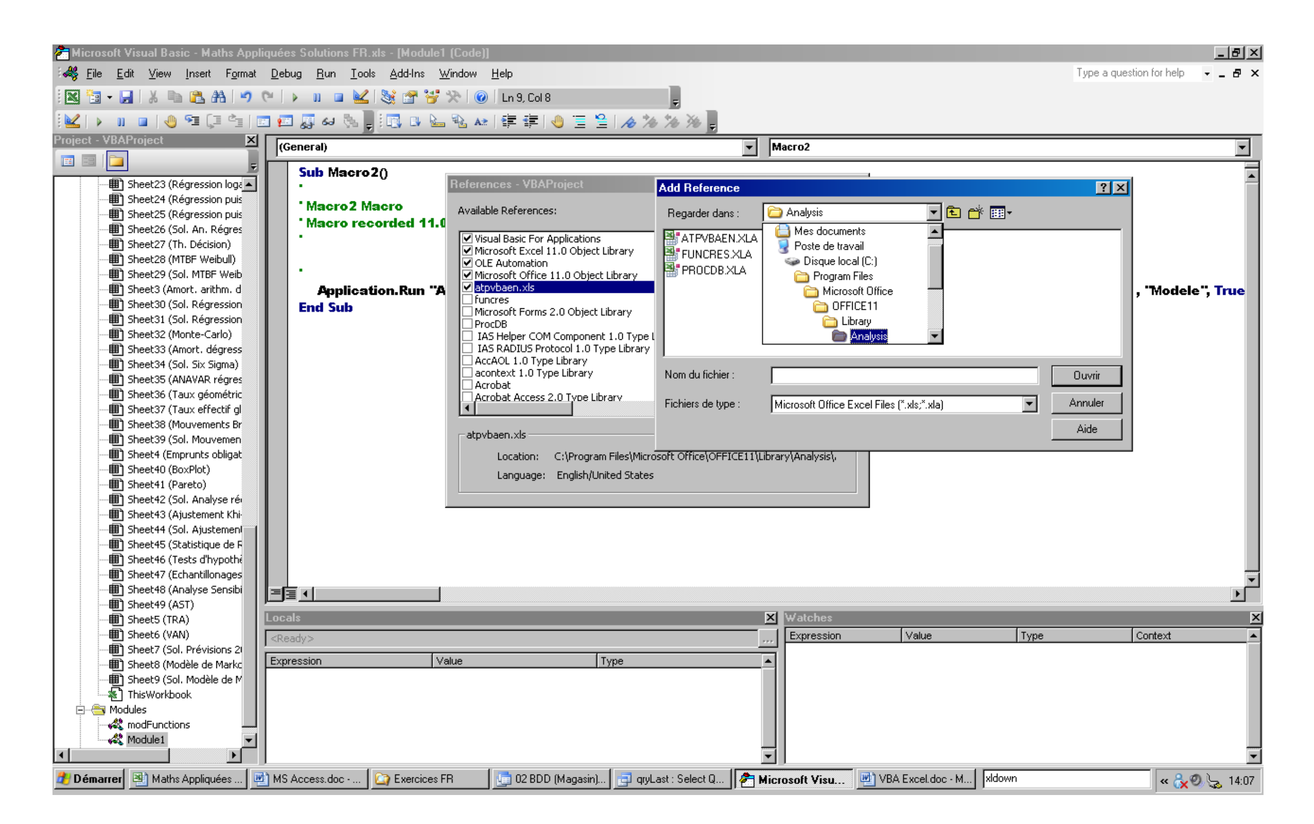

10.11.2 Appel de l'utilitaire d'analyse de la régression

L'enregistrement d'une macro avec l'utilitaire d'analyse faisant usage de la régression linéaire fonctionne **mais pas son exécution**!!!

Pour lancer l'utilitaire d'analyse de la régression, le code est simplement enregistré par la macro:

```
Sub AnalyseRegression()
     Application.Run "ATPVBAEN.XLA!Regress", ActiveSheet.Range("G6:G135"),
ActiveSheet.Range("H6:K135"), False, True, , "Modele", True, False, False,
False, , False
End Sub
```

Cela vient du fait qu'il faut installer la référence ATPVBAEN.XLA dans l'éditeur VBAE comme indiqué ci-dessous:



et ensuite il peut arriver qu'il failler ferme MS Excel et le rouvrir!



11. CLASSES

La programmation par classes (POO) permet uniquement de mieux structurer le code. Effectivement à notre connaissance il est toujours possible de toute faire avec des fonctions et procédures seulement mais le code n'est dès lors pas toujours aussi facile à relire si l'on compare aux relectures du monde qui nous environne et qui est constitué d'objects, de propriétés, de méthodes et d'instances imbriquées ou non.

Considérons comme exemple un code de classe permettant de manipuler des tableaux:

Class Modules
Matrix

```
Option Explicit

'attribut privé représentant une matrice
Private mat() As Double

'méthode qui initialise la matrice (elle est remplie avec des zéros)
'nrow est le nombre de lignes et ncol le nombre de colonnes
Public Sub construct(nrow, ncol)
    Dim i As Integer
    Dim j As Integer

  'Redéfinit la taille du tableau mat
    ReDim mat(nrow - 1, ncol - 1)

    'Boucle sur les éléments de la matrice pour la remplir de zéros
    For i = 0 To nrow - 1
        For j = 0 To ncol - 1
            mat(i, j) = 0
        Next j
    Next i
End Sub

'méthode qui initialise la matrice avec des valeurs (tbl() est un
ParamArray qui récupère la liste de valeur)
'nrow est le nombre de lignes et ncol le nombre de colonnes
Public Sub construct_with_values(nrow, ncol, ParamArray tbl() As Variant)
    Dim i As Integer
    Dim j As Integer

    'Redéfinit la taille du tableau
    ReDim mat(nrow - 1, ncol - 1)

    'Boucle sur les éléments de la matrice pour le remplir
    For i = 0 To nrow - 1
        For j = 0 To ncol - 1
            mat(i, j) = tbl(i * ncol + j)
        Next j
    Next i
End Sub

'le Let attribue la valeur value à l'élément (i,j) de la matrice
Property Let item(i, j, value)
   mat(i, j) = value
End Property
```

```
'le get restitue l'élément (i,j) de la matrice
Property Get item(i, j)
    item = mat(i, j)
End Property

'méthode qui ajoute une ligne à la fin de la matrice
Public Sub add_row(ParamArray row())
    Dim i As Integer
    Dim j As Integer
    Dim tmpmat() As Double

    ReDim tmpmat(UBound(mat, 1) + 1, UBound(mat, 2))
    For i = 0 To UBound(mat, 1)
        For j = 0 To UBound(mat, 2)
            tmpmat(i, j) = mat(i, j)
        Next j
    Next i

    For j = 0 To UBound(row)
        tmpmat(UBound(mat, 1) + 1, j) = row(j)
    Next j

    mat = tmpmat
End Sub

'méthode qui calcule la somme d'une ligne
Public Function row_sum(i) As Double
    Dim j As Integer

    row_sum = 0
    For j = 0 To UBound(mat)
        row_sum = row_sum + mat(i, j)
    Next j
End Function

'méthode qui calcule la somme de la matrice courante et de la matrice
passée en argument
Public Function plus(m As Matrix) As Matrix
    Dim i As Integer, j As Integer
    Set plus = New Matrix


      'c'est le fait que plus soit une fonction dans la classe qui permet
d'en utiliser les méthodes et constructeurs et autres… de même c'est le
fait que m soit un paramètre de la fonction qui permet aussi de faire de
même avec!
    Call plus.construct(UBound(mat, 1) + 1, UBound(mat, 2) + 1)

    For i = 0 To UBound(mat, 1)
        For j = 0 To UBound(mat, 2)
            plus.item(i, j) = mat(i, j) + m.item(i, j)
        Next j
    Next i

End Function

'méthode qui affiche la matrice dans la fenêtre de débogage
Public Sub debug_print()
    Dim str_row As String, i As Integer, j As Integer

    For i = 0 To UBound(mat, 1)
```

```
        str_row = ""
        For j = 0 To UBound(mat, 2)
            str_row = str_row & "    " & mat(i, j)
        Next j
        Debug.Print str_row
    Next i
End Sub
```

Ensuite on peut tester tout cette classe pour voir que cela se comporte bien:

```
Sub test()
    Dim A As Matrix, B As Matrix, C As Matrix

    Set A = New Matrix
    Set B = New Matrix

    Call A.construct_with_values(2, 3, 3, 2, 5, 6, 7, 8)
    Call B.construct_with_values(2, 3, -4, -7, 9, -1, -6, 0)

    Debug.Print "Matrice A :"
    A.debug_print
    Debug.Print "Matrice B :"
    B.debug_print

    Set C = A.plus(B)

    Debug.Print "Matrice A + B :"
    C.debug_print

End Sub
```


Ce qui va donner:

```
Immediate
Matrice A :
    3    2    5
    6    7    8
Matrice B :
    -4    -7    9
    -1    -6    0
Matrice A + B :
    -1    -5    14
    5    1    8
```

12. GESTION DES ERREURS AVANCÉE

Nous avons rencontré à plusieurs reprises jusqu'ici les bases de la gestion des erreurs mais souvent avec une gestion basique.

Allonsu un peu plus loin avec l'exemple suivant:

```
Public Function DiviseDeux(nb1 As Variant, nb2 As Variant)

    'Nous mettons en Variant afin de laisser passer les lettres au delà de
la fonction

    If IsNumeric(nb1) = False Or IsNumeric(nb2) = False Then
    'a nouveau nous pourrions écrire: not(isnumeric(nb1)) or
not(isnumeric(nb2))
        MsgBox "Une des deux entrées n'est pas un nombre", vbCritical +
vbRetryCancel, "Attention!"
    End If

On Error GoTo GestionErreurs
    divisedeux = nb1 / nb2
    'N'oubliez pas de quitter la fonction sinon quoi la gestion des erreurs
va être executée
    Exit Function
    'Ou Exit Sub dans le cas d'une routine

GestionErreurs:
    MsgBox Str(Err.Number) & ": " & Err.Description, , "Erreur"
    'le message à renvoyer dans la cellule en cas d'erreur
    nb2=inputbox("Nouvelle valeur pour nb2")
    'Vous pouvez écrire ensuite (pour continuer avec autre chose):
    'Call NomAutreProcedureAExecuter
    'ou
    Resume 'revient à la ligne qui a provoqué l'erreur

End Function
```


Attention!!! Il arrive relativement souvent que les messages d'erreurs par défaut de VBA n'ont rien à voir avec le problème réel. Donc il faut serrer les dents et parfois avoir beaucoup de patience…

12.1 Forcer la génération d'erreurs (erreurs non interceptées)

Pour tester des codes ou "officialiser le format d'affichage d'une erreur" il peut être utile par endroits de générer des erreurs. La génération des erreurs est aussi utile bien évidemment pour les erreurs non interceptées par Microsoft. Pour ce faire il suffit d'ajouter à l'endroit voulu (voir en rouge ci-dessous):

```
Public Function DiviseDeux(nb1 As Variant, nb2 As Variant)

    On Error GoTo GestionErreurs
    'Nous mettons en Variant afin de laisser passer les lettres au delà de
la fonction

    If IsNumeric(nb1) = False Or IsNumeric(nb2) = False Then
```

```
    'a nouveau nous pourrions écrire: not(isnumeric(nb1)) or
not(isnumeric(nb2))
        Err.Raise Number:=007, Description:="Une des deux entrées n'est pas
un nombre", vbCritical + vbRetryCancel, "Attention!"
    End If

    divisedeux = nb1 / nb2
    'N'oubliez pas de quitter la fonction sinon quoi la gestion des erreurs
va être executée
    Exit Function
    'Ou Exit Sub dans le cas d'une routine

GestionErreurs:
    MsgBox Str(Err.Number) & ": " & Err.Description, , "Erreur"
    'le message à renvoyer dans la cellule en cas d'erreur
    nb2=inputbox("Nouvelle valeur pour nb2")
    'Vous pouvez écrire ensuite (pour continuer avec autre chose):
    'Call NomAutreProcedureAExecuter
    'ou
    Resume 'revient à la ligne qui a provoqué l'erreur

End Function
```

12.2 Numéroter les lignes et renvoyer le numéro de ligne d'erreur

Si nous numérotons les lignes à la main (ou avec un add-in adéquat), il est possible lors d'une erreur de renvoyer le numéro de la ligne qui a posé le problème. Voici un exemple:


```
Sub EurreurLigneDemo()
    Dim i As Long

    On Error GoTo GestionErreur

10  Debug.Print "A"
20  Debug.Print "B"
30  i = "Sid"
40  Debug.Print "A"

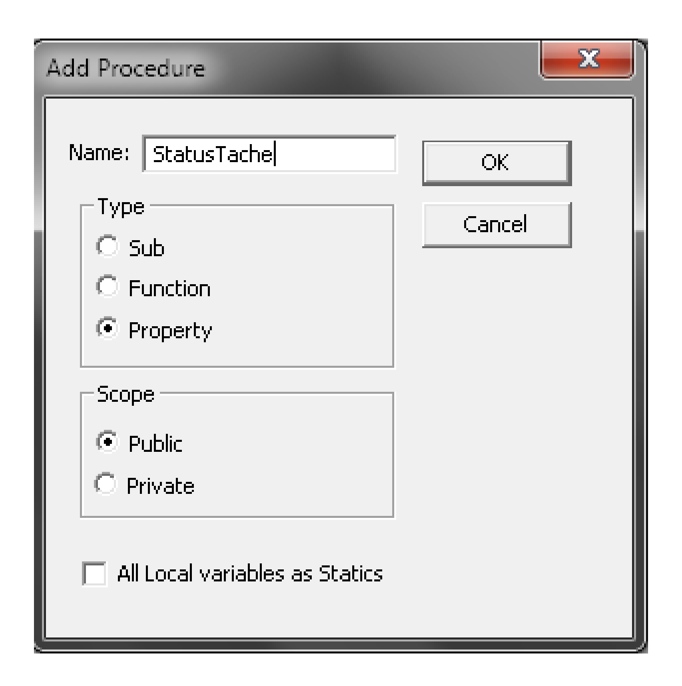
50  Exit Sub

    GestionErreur:
        MsgBox "Erreur sur la ligne : " & Erl
End Sub
```

13. PROPRIÉTÉS

Le but d'une propriété est de ranger proprement une procédure utilisée régulièrement et renvoyant toujours le même type de valeur. Évidemment, la même chose peut être faite avec une simple routine mais il ne faut pas oublier que faire un code qui marche et un code propre qui marche sont deux choses distinctes.

Donc dans V.B.E. si nous créons un nouvel élément de code, nous avons:



Nous avons alors:

```
Public Property Get StatusTache() As Variant

End Property
```

```
Public Property Let StatusTache(ByVal vNewValue As Variant)

End Property
```

L'instruction *Property Get* a pour effet de renvoyer l'information à la procédure ou routine qui appelle et l'instruction *Property Let* effectue directement l'action elle-même. Voyons un exemple très simple qui suffira à comprendre les tenants et aboutissants:

```
Sub PrioriteTache()
    Dim intPriorite As Integer
    intPriorite = 300
    MsgBox StatusTache(intPriorite)
End Sub

Public Property Get StatusTache(intPriorite) As Variant
    Select Case intPriorite
    Case Is < 100
        StatusTache = "Faible"
    Case 100 To 451
```

```
        StatusTache = "Moyen"
    Case 451 To 550
        StatusTache = "A faire"
    Case Is > 551
        StatusTache = "Urgent"
    End Select
End Property
```

Voici un exemple plus complet repris de l'aide du logiciel lui-même:

```
Dim CurrentColor As Integer
Const BLACK = 0, RED = 1, GREEN = 2, BLUE = 3

' Set the pen color property for a Drawing package.
' The module-level variable CurrentColor is set to
' a numeric value that identifies the color used for drawing.

Property Let PenColor(ColorName As String)
    Select Case ColorName ' Check color name string.
    Case "Red"
        CurrentColor = RED ' Assign value for Red.
    Case "Green"
        CurrentColor = GREEN ' Assign value for Green.
    Case "Blue"
        CurrentColor = BLUE ' Assign value for Blue.
    Case Else
        CurrentColor = BLACK ' Assign default value.
    End Select
End Property

' Returns the current color of the pen as a string.

Property Get PenColor() As String
    Select Case CurrentColor
    Case RED
        PenColor = "Red"
    Case GREEN
        PenColor = "Green"
    Case BLUE
        PenColor = "Blue"
    End Select
End Property

Sub pen()
' The following code sets the PenColor property for a drawing package
' by calling the Property Let procedure.
PenColor = "Red"

' The following code gets the color of the pen
' calling the Property Get procedure.
ColorName = PenColor

MsgBox ColorName

End Sub
```

14. SHAPES

Un cas super pratique dans MS Excel (ou dans certains autres logiciels de la suite MS Office) consiste à créer une petite forme automatique rectangulaires indiquant à l'utilisateur que les calculs sont en cours.

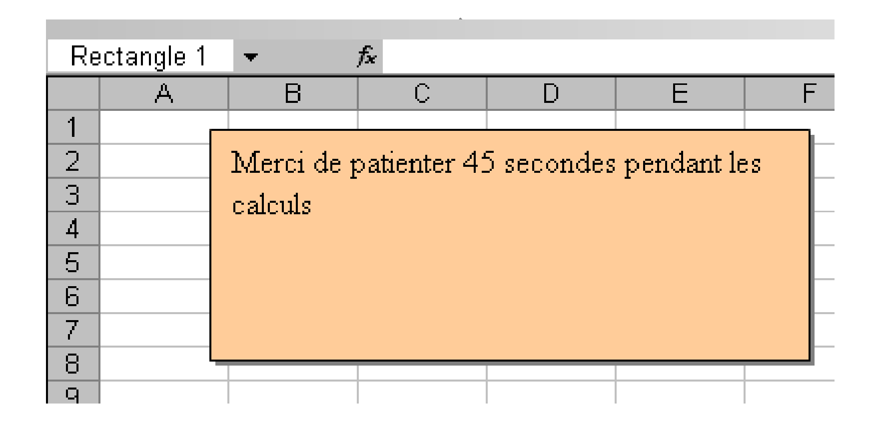
Effectivmeent, les MsgBox ne sont pas adaptées à cette situation car elles arrêtent le code lors de leur affichage. La solution consiste alors à créer une forme du type suivant:

Merci de patienter 45 secondes pendant les calculs

et ensuite dans n'importe quelle partie du code on peut la masquer où l'afficher en utilisant:

```
Sheet1.Shapes("Rectangle 1").Visible
```

où le nom de la forme peut être obtenue en utilisant:



15. USERFORMS

La commande msgbox et inputbox de VB sont souvent insuffisant en matière d'interacitivé avec l'utilisateur ou même de personnalisation pour le développeur. Certes avec InputBox ou MsgBox on peut même capture des plages comme le montre l'exemple ci-dessous:

```
Sub Interaction()

   Dim rng As Range

   Set rng = Application.InputBox("Please select range you wish to send.", 
Type:=8, Default:=Application.Selection.Address)

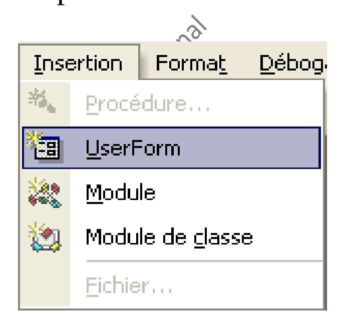
   MsgBox "Plage sélectionnée: [(" & rng.Cells.Row & "," & rng.Cells.Column
& "),(" & rng.Cells.Rows.Count + rng.Cells.Row - 1 & "..."

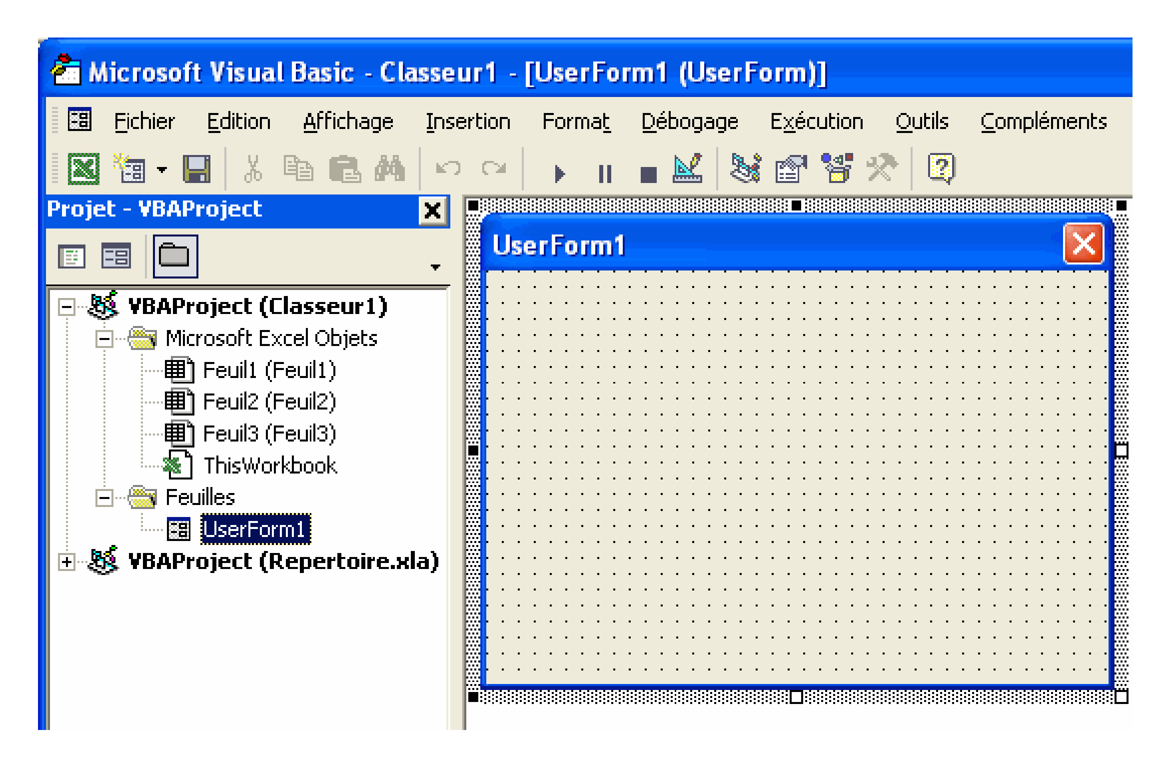
End Sub
```

Il faut alors passer par l'utilisation de formulaires plus élaborés que l'on nomme les "userform".

Les UserForm sont des boîtes de dialogues personnalisées, offrant une interface intuitive entre l'application et l'utilisateur.

Sous VBAE, les UserForm sont créés par le menu **Insertion/UserForm**.

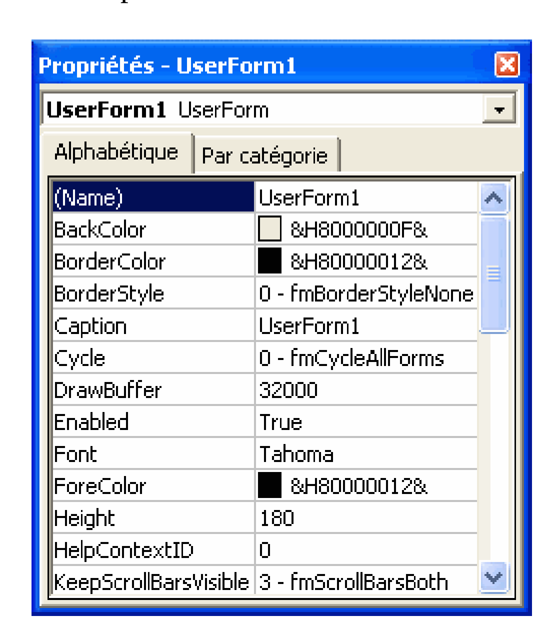




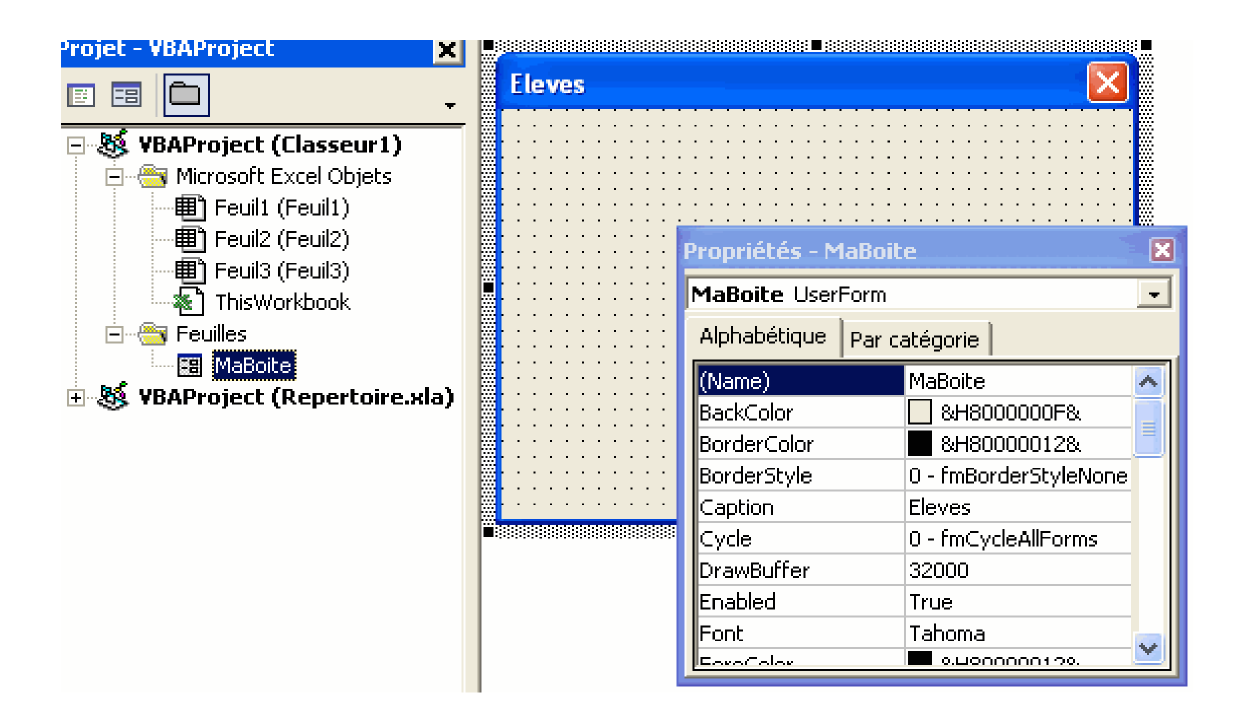
Par défaut, les UserForm sont nommés "UserForm1", "UserForm2" ...

Chaque UserForm possède ses propres propriétés tel que son nom, sa couleur, sa taille, sa position ...

Les propriétés d'un UserForm s'affichent en cliquant sur l'icône , par le menu **Affichage/Fenêtre Propriétés** ou par la touche **F4**.



La propriété `Name` change le nom de l'UserForm, la propriété `Caption`, son titre.



Les propriétés permettent de personnaliser les UserForm. Vous pouvez changer la couleur de fond par la propriété `BackColor`, ajouter une bordure par la propriété `BorderStyle`, définir sa couleur par la propriété `BorderColor`, mettre une image de fond par la propriété `Picture`...

Le dimensionnement d'un UserForm peut se faire avec la souris ou en définissant sa taille par ses propriétés `Width` (Largeur) et `Height` (Hauteur).

15.1 Charger un formulaire à l'ouverture du fichier Excel

Pour charger un formulaire à l'ouverture du fichier Excel on utilisera la procédure événementielle Workbook_Open the l'objet ThisWorkbook. Nous avons alors typiquement:

```
Private Sub Workbook_Open()
    Nom_Userform.Show
End Sub
```

ou pour pouvoir cliquer derrière:

```
Private Sub Workbook_Open()
    Nom_Userform.Show vbModeless
End Sub
```

ou en masquant l'ensemble de l'interface Excel:

```
Private Sub Workbook_Open()
    Application.Visibile=False
    Nom_Userform.Show
End Sub
```

15.2 Événements sur formulaires

Citons l'événement suivant qui est souvent demandé par ceux qui créent des contrôles de sécurité (cela désactive la possibilité de ferme le formulaire):

```
Private Sub UserForm_QueryClose (Cancel As Integer, CloseMode As Integer)
    If CloseMode = vbFormControlMenu Then
        MsgBox "Clicking the Close button does not work."
        Cancel = True
    End If
End Sub
```

ou encore le code suivant qui permet de mettre par exemple un formulaire noir sur tout l'écran afin de masquer complétement MS Excel (cela donne un chouette effet pour certaines applications):

```
Private Sub UserForm_Activate()
    With Me
        .Height = Application.Height
        .Width = Application.Width
        .Left = Application.Left
        .Top = Application.Top
    End With
End Sub
```

15.3 Redimensionner un formulaire

Parfois il est nécessaire de faire en sorte que l'utilisateur puisse redimensionner un UserForm alors qu'il n'existe aucune propriété pour cela dans le V.B.A. de la suite Office.

Dans ce type de situation on a souvent la propriété ScrollBars du formulaire qui sera à la valeur fmScrollBarsBoth.

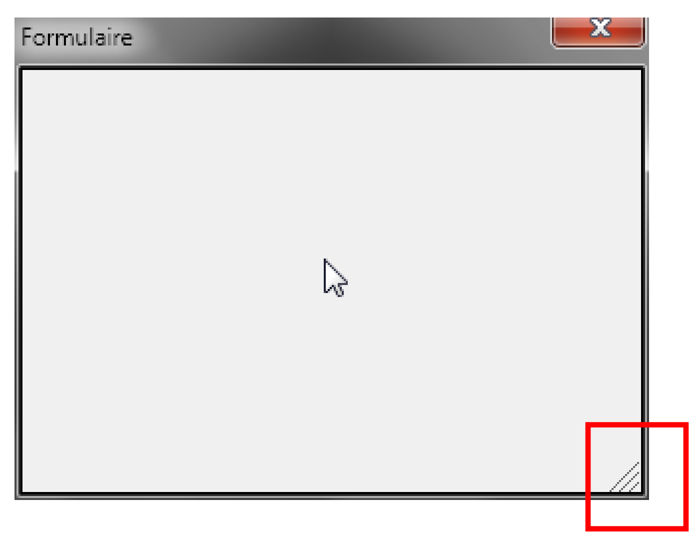
L'astuce consiste alors dans un premier temps d'écrire le code (astucieux!) à l'initialisation du formulaire:

```
Private Const MResizer = "ResizeGrab"
Private WithEvents objResizer As MSForms.Label
Private LeftResizePos As Single
Private TopResizePos As Single

Private Sub UserForm_Initialize()
    'Add a resizing control to bottom right corner of UserForm
    Set objResizer = Me.Controls.Add("Forms.label.1", MResizer, True)
    With objResizer
        .Caption = Chr(111)
        .Font.Name = "Marlett"
        .Font.Charset = 2
        .Font.Size = 14
        .BackStyle = fmBackStyleTransparent
        .AutoSize = True
        .ForeColor = RGB(100, 100, 100)
        .MousePointer = fmMousePointerSizeNWSE
        .ZOrder
        .Top = Me.InsideHeight - .Height
        .Left = Me.InsideWidth - .Width
    End With
```

```
End Sub
```

Ce qui donnera dans le coin du formulaire:



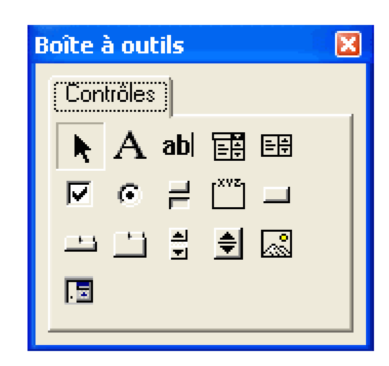
Ensuite, il suffit d'ajouter dans le code du formulaire:

```
Private Sub objResizer_MouseMove(ByVal Button As Integer, ByVal Shift As
Integer, ByVal X As Single, ByVal Y As Single)
    If Button = 1 Then
        With objResizer
            .Move .Left + X - LeftResizePos, .Top + Y - TopResizePos
            Me.Width = Me.Width + X - LeftResizePos
            Me.Height = Me.Height + Y - TopResizePos
            .Left = Me.InsideWidth - .Width
            .Top = Me.InsideHeight - .Height
        End With
    End If
End Sub
```


Le formulaire est dès lors redimensionnable!

15.4 Contrôles sur formulaires

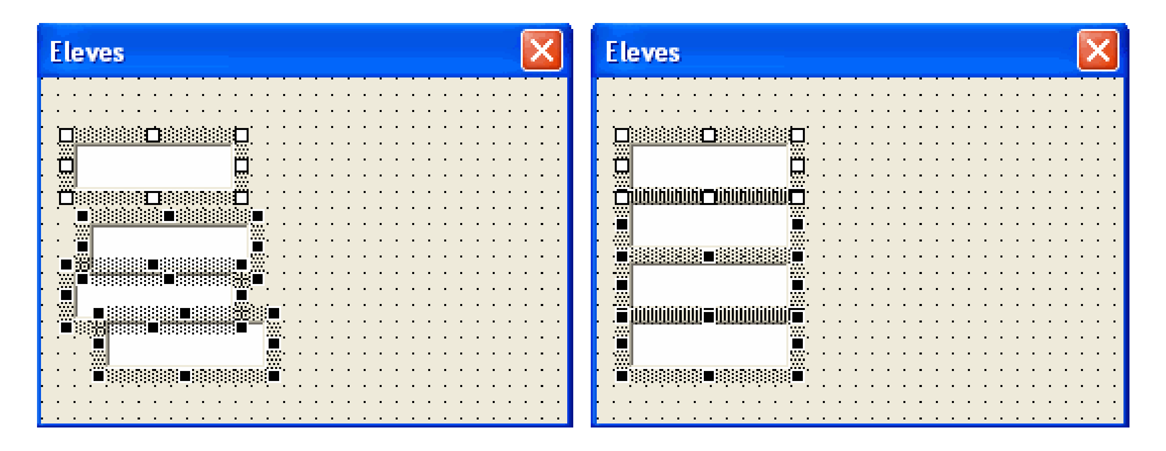
Chaque UserForm va recevoir des contrôles. En cliquant sur le UserForm, une boite à outils doit apparaître. Si ce n'est pas le cas, affichez-la en cliquant sur l'icône ou par le menu **Affichage/Boîte à outils**.



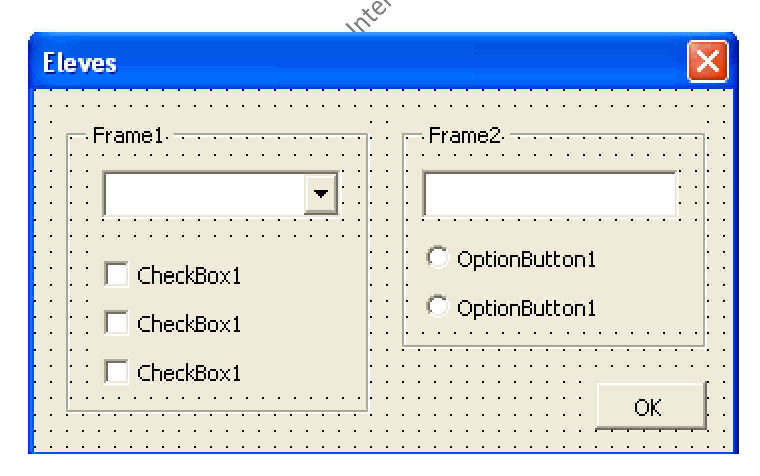
Pour ajouter un contrôle sur le UserForm, vous pouvez soit cliquer sur le contrôle désiré puis, sur le UserForm, tracer un rectangle qui définira sa taille ou simplement faire un cliquer-glisser du contrôle sur l'UserForm.

Les UserForm possèdent une grille matérialisée par des points. Elle permet l'alignement des contrôles. Vous pouvez la masquer, la désactiver ou définir sa taille par le menu **Outils/Options** dans l'onglet **Général**.

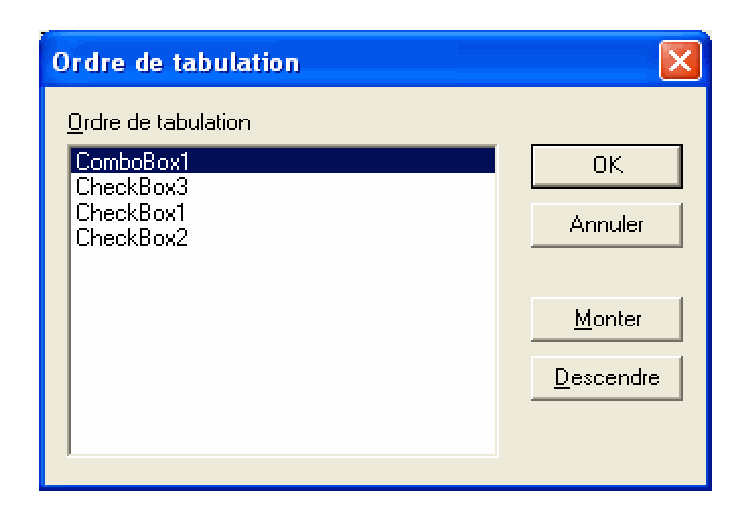
Le menu **Format** de V.B.E. permet d'aligner les contrôles. Par exemple le menu **Format/Aligner/Gauche** puis le menu **Espacement/Vertical/Egaliser** permet un alignement régulier des contrôles:



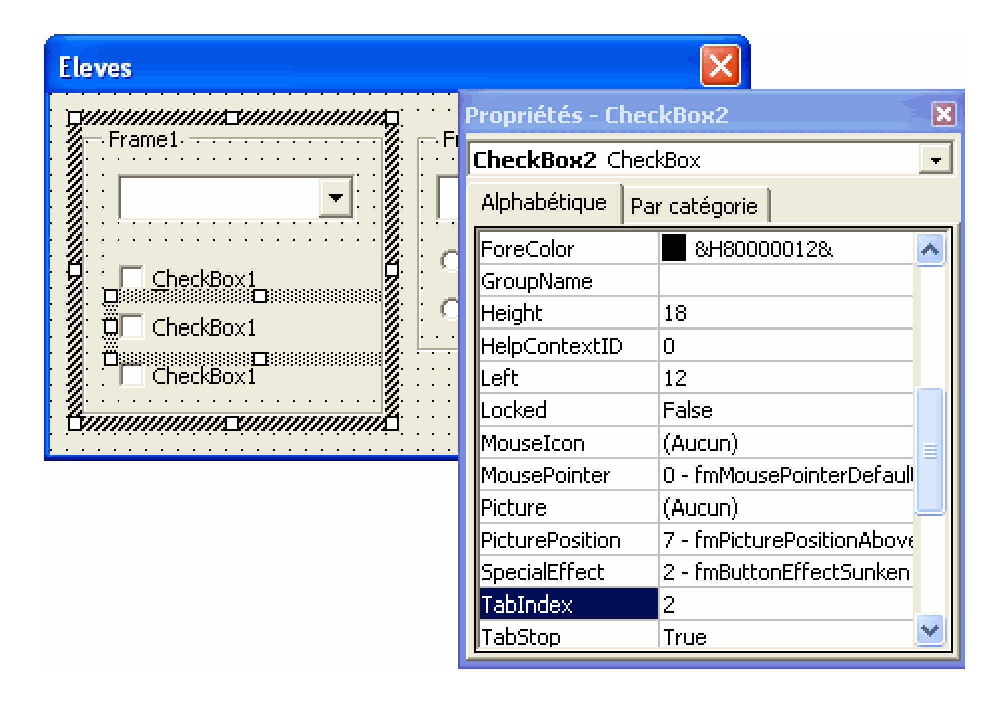
Le contrôle *Frame* permet de grouper des contrôles.



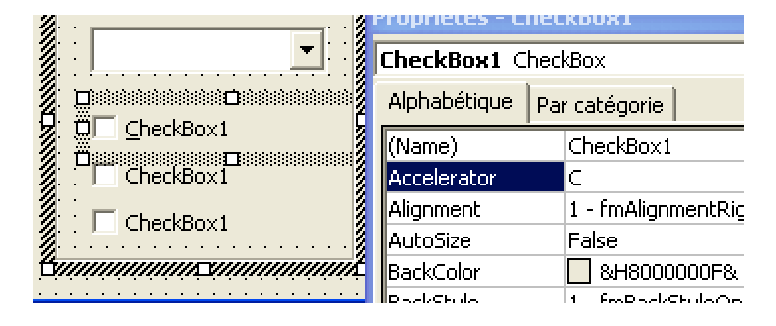
Le UserForm doit permettre à l'utilisateur de passer d'un contrôle à l'autre par la touche **Tabulation** de façon ordrée. Le menu **Affichage/Ordre de tabulation** permet de paramétrer l'ordre de tabulation. Cliquez sur l'UserForm pour changer l'ordre des deux frames et du bouton **OK** et sélectionnez une frame pour changer l'ordre des contrôles qu'elle contient.



Vous pouvez également changer l'ordre de tabulation par la propriété "TabIndex" de chaque contrôle.

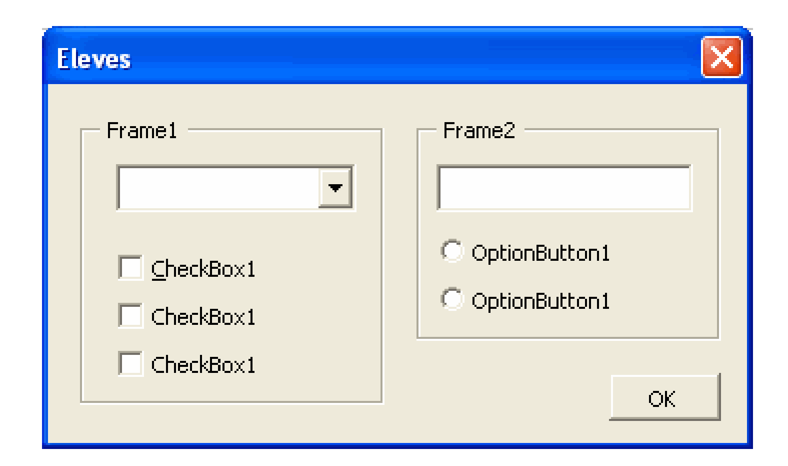


Vous pouvez affecter une touche de raccourci "**Alt**+caractère" à un contrôle par sa propriété **Accelerator**. Utilisez un caractère du nom du contrôle, celui-ci sera souligné, indiquant à l'utilisateur quelle touche de raccourci utiliser:



L'affichage des UserForm s'effectue par la méthode *Show* de l'UserForm. Cette instruction doit être placée à l'intérieur d'une procédure dans un module.

```
Sub AfficheUF()
    MaBoite.Show
    'MaBoite.Repaint pour faire un refresh sans avoir à fermer/rouvrir
    'MaBoite.Repaint vbModeless pour pouvoir écrire derrière la boîte de
dialogue
End Sub
```



Par défaut, un UserForm est modal, c'est à dire que l'utilisateur ne peut effectuer aucune action sur l'application tant qu'il n'est pas fermé. Depuis la version 2000 d'Excel, il est possible d'afficher des boites non modales, permettant l'utilisation des feuilles de calcul en gardant le UserForm affichée. La syntaxe est:

```
Sub AfficheUF()
    MaBoite.Show 0 '0= vbModeless
End Sub
```


Remarques:

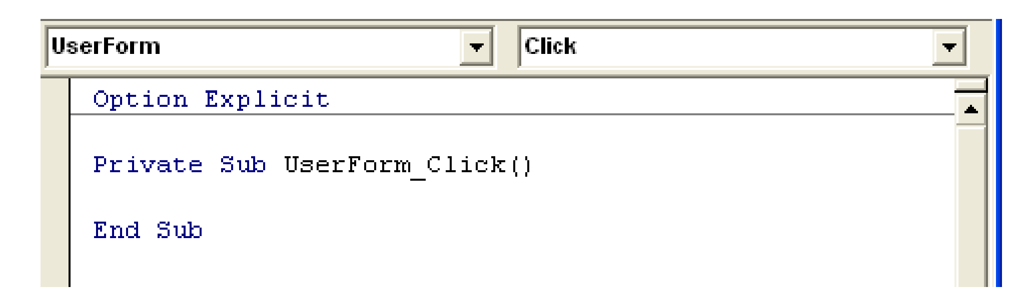
R1. L'instruction *Load* charge le UserForm en mémoire sans l'afficher.

R2. L'instruction *Unload* ferme le UserForm en le déchargeant de la mémoire. La syntaxe de cette instruction est: Unload UserForm.

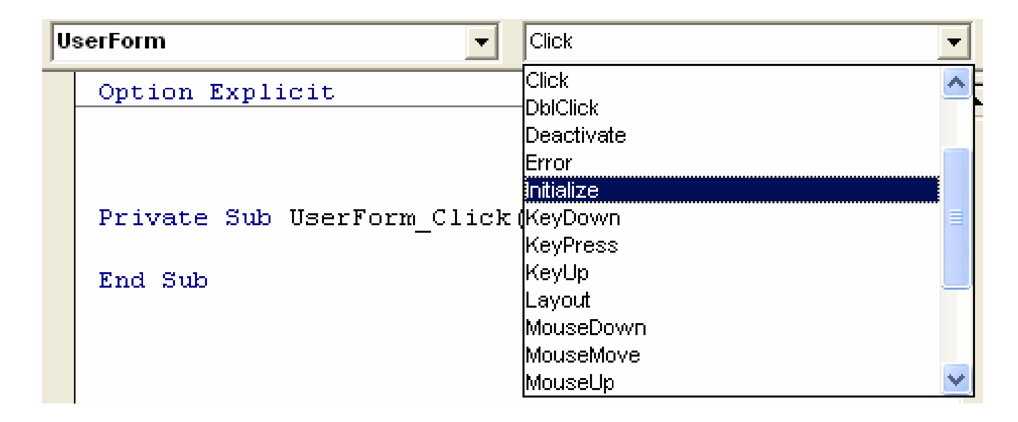
R3. Il est également possible de fermer un UserForm en gardant en mémoire la valeur de ses contrôles par la méthode *Hide*. La syntaxe devient: UserForm.Hide.

Chaque UserForm possède son propre module.

Pour y accéder, cliquez sur le UserForm ou sur un contrôle puis tapez "F7" ou faîtes un double-clic sur l'objet. Par défaut, le module s'affichera avec une procédure évènementielle de type privée de l'objet sélectionné.

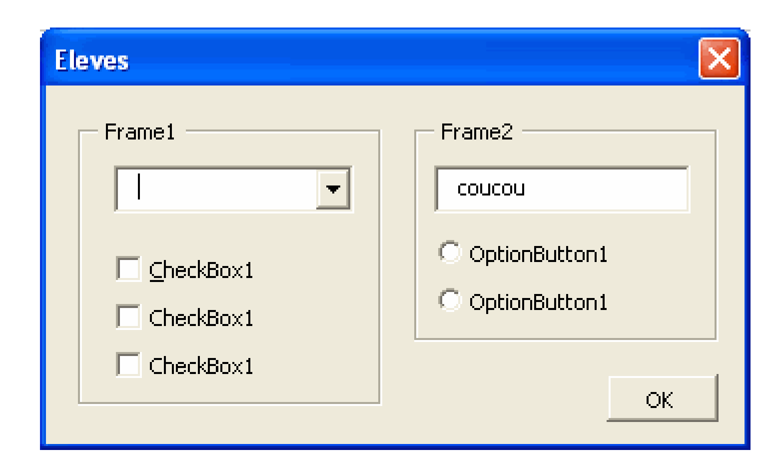


Les deux listes déroulantes en haut du module permettent de sélectionner l'objet et son évènement.



Dans cet exemple, la procédure Initialize de l'objet UserForm va être créée et ses instructions vont être exécutées au chargement de la boite.

```
Private Sub UserForm_Initialize()
    TextBox1 = "coucou"
End Sub
```



Si l'évènement Initialize se produit au chargement d'un UserForm, l'évènement QueryClose se produit à sa fermeture. Dans l'exemple suivant, un message contenant le texte de l'objet TextBox1 s'affichera à la fermeture de la boite.

```
Sub UserForm_QueryClose(Cancel As Integer, CloseMode As Integer)
    Msgbox TextBox1
End Sub
```

L'élément Cancel de l'évènement QueryClose invalide la fermeture de la boite si sa valeur est 1 et l'élément CloseMode défini la manière dont la boite cherche à être fermée. Si l'utilisateur cherche à la fermer en cliquant sur la croix, CloseMode prend comme valeur 0, sinon CloseMode prend comme valeur 1.

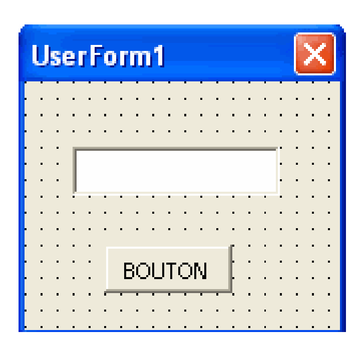
L'exemple suivant montre comment obliger l'utilisateur à fermer la boite en cliquant sur le bouton "OK":

```
Private Sub CommandButton1_Click()
```

```
    Unload MaBoite
End Sub

PrivateSub UserForm_QueryClose(Cancel As Integer, CloseMode As Integer)
    If CloseMode = 0 Then 'Si on clique sur la croix
    'Ce qui peut s'écrir aussi CloseMode=vbFormControlMenu
        Msgbox "Fermez la boite avec le bouton OK"
        Cancel = True 'Invalide la fermeture
    End If
End Sub
```

Les contrôles possèdent la propriété Visible qui permet de les rendre visible ou invisible. Dans l'exemple suivant, le click sur BOUTON va masquer ou afficher la zone de texte.

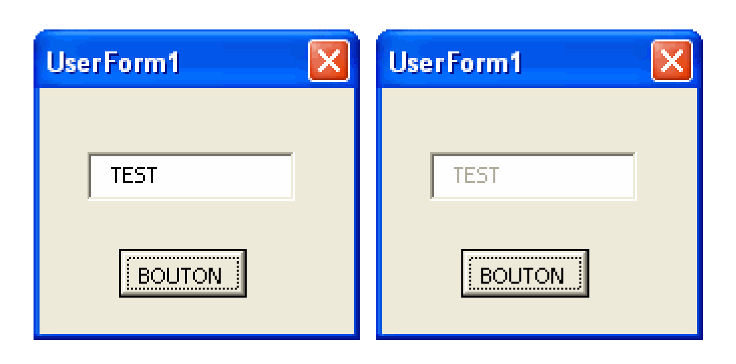



```
Private Sub CommandButton1_Click()
    If TextBox1.Visible = True Then
        TextBox1.Visible = False
    Else
        TextBox1.Visible = True
    End If
End Sub
```

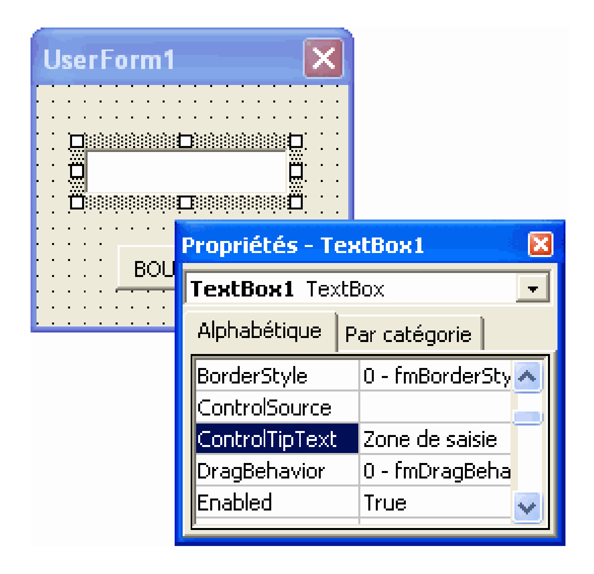

Les contrôles possèdent la propriété Enabled qui peut interdire son accès à l'utilisateur. Lorsqu'on contrôle n'est pas accessible, son aspect change.

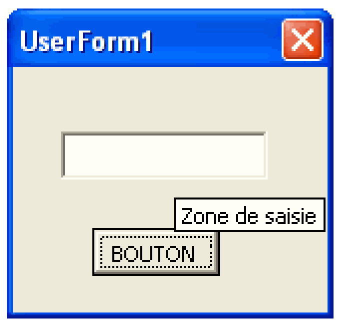
Dans l'exemple suivant, le click sur BOUTON va interdire ou rendre accessible l'accès à la zone de texte.

```
Private Sub CommandButton1_Click()
    If TextBox1.Enabled = True Then
        TextBox1.Enabled = False
    Else
        TextBox1.Enabled = True
    End If
End Sub
```



Les contrôles possèdent la propriété *ControlTipText* qui affiche une étiquette lors du survol de la souris.





Les contrôles de type *TextBox* ont une propriété *AutoWordSelect* qui détermine la façon dont le texte est sélectionné dans la zone d'édition. Si la propriété est sur *True*, l'utilisateur étend la sélection au-delà d'un mot, le mot entier est sélectionné et la sélection se fait ensuite mot par mot. Si la propriété est sur *False*, la sélection se fait caractère par caractère.

Il est possible de créer des éléments de formulaires via V.B.A. Dans l'exemple suivant, le clic sur BOUTON va créer un nouveau bouton nommé CommandButton2 et contenant le Caption "This is fun" avec le nom du bouton concaténé:

```
Dim Mycmd As Control

Private Sub CommandButton1_Click()

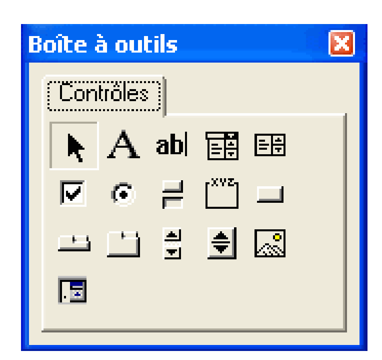
    Set Mycmd = Controls.Add("Forms.CommandButton.1", CommandButton2,
Visible)
    Mycmd.Left = 18
    Mycmd.Top = 150
    Mycmd.Width = 175
    Mycmd.Height = 20
    Mycmd.Caption = "This is fun." & Mycmd.Name

End Sub
```

Les premiers arguments de la commande Controls.Add peuvent être choisi selon la liste ci-dessous:

CheckBox	Forms.CheckBox.1
ComboBox	Forms.ComboBox.1
CommandButton	Forms.CommandButton.1
Frame	Forms.Frame.1
Image	Forms.Image.1
Label	Forms.Label.1
ListBox	Forms.ListBox.1
MultiPage	Forms.MultiPage.1
OptionButton	Forms.OptionButton.1
ScrollBar	Forms.ScrollBar.1
SpinButton	Forms.SpinButton.1
TabStrip	Forms.TabStrip.1
TextBox	Forms.TextBox.1
ToggleButton	Forms.ToggleButton.1

La boite à outils affiche les contrôles standards de V.B.A.

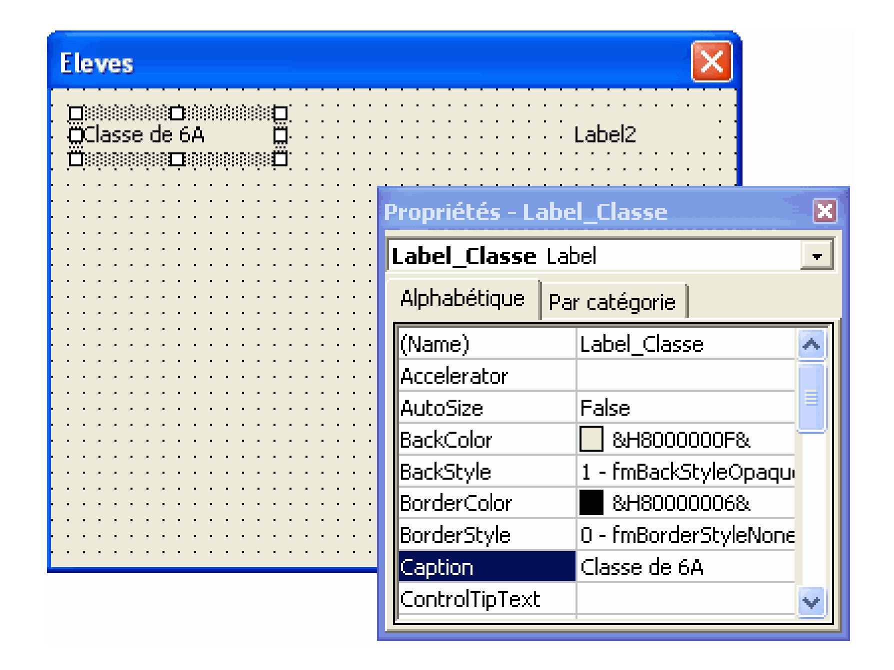


15.4.1 Sélection

Cet outil permet de sélectionner, de déplacer et de redimensionner les contrôles créés sur l'UserForm.

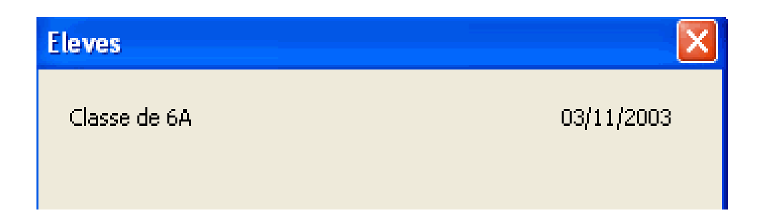
15.4.2 Label ou étiquette

Cet outil permet de créer une zone de texte non modifiable par l'utilisateur.



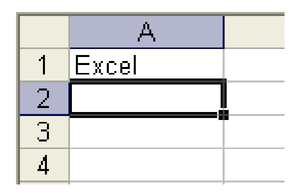
Dans cet exemple, 2 étiquettes ont été créées. Par défaut leur nom était Label1 et Label2. Pour plus de confort, elles ont été renommées Label_Classe et Label_Date. La valeur de Label_Class étant fixe, elle a été saisie dans sa propriété Caption. La valeur de Label_Date étant variable, elle peut être définie dans l'évènement Initialize de l'UserForm (renommé MaBoite).

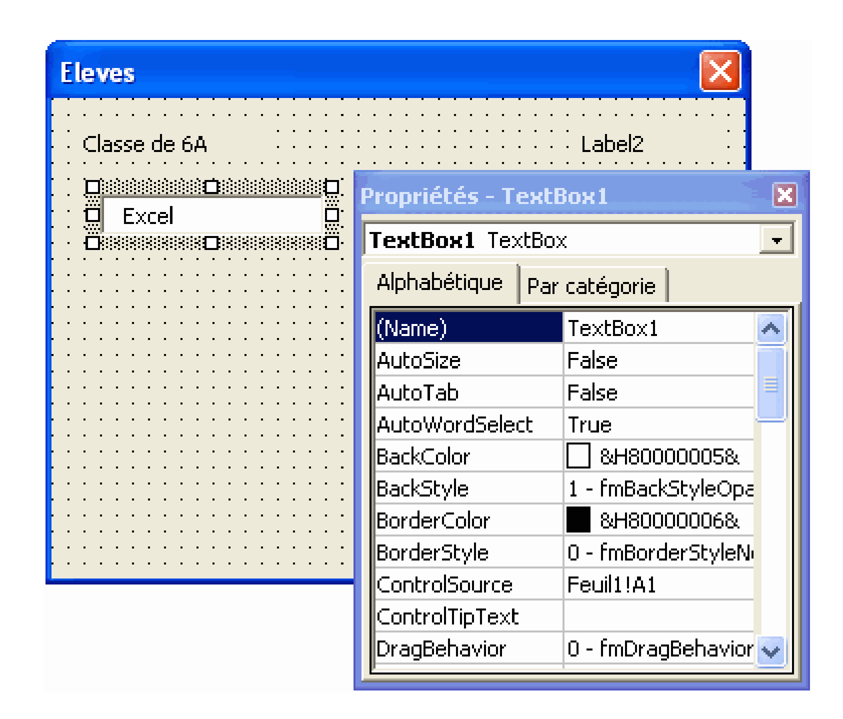
```
Private Sub UserForm_Initialize()
    Label_Date.Caption = Date
End Sub
```

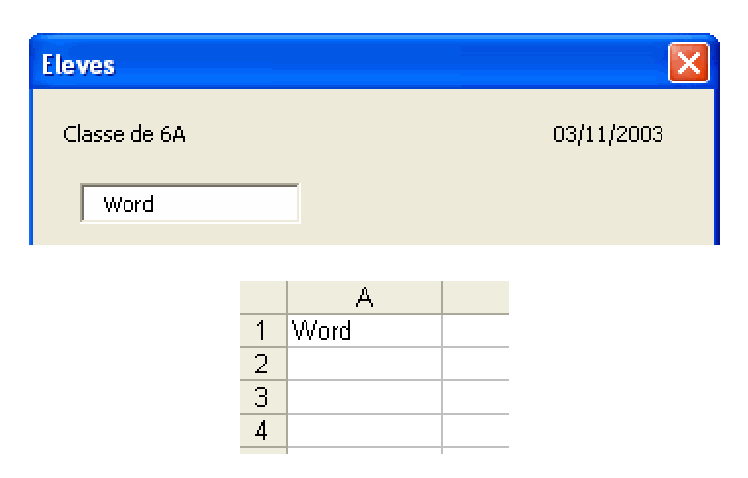
15.4.3 TextBox ou zone de texte

ab| Cet outil permet de créer une zone de texte pouvant être saisie ou modifiée par l'utilisateur. Une zone de texte peut faire référence à une cellule par la propriété ControlSource.



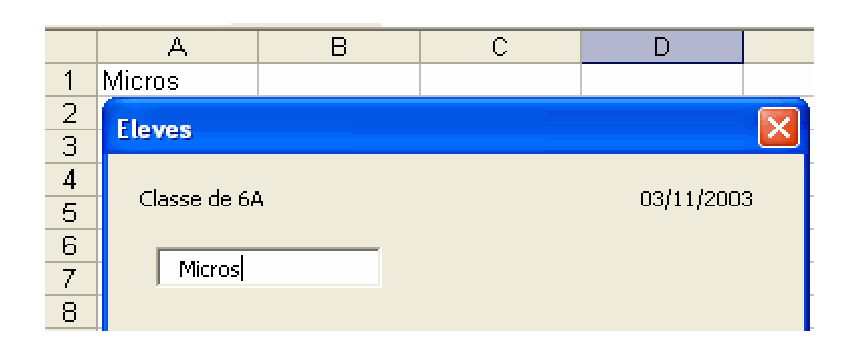


Si l'utilisateur modifie la zone de texte, la valeur de la cellule A1 sera modifiée.



La valeur de la cellule A1 peut également prendre la valeur de la zone de texte par une procédure événementielle TextBox_Change.

```
Private Sub TextBox1_Change()
    Range("A1") = TextBox1
End Sub
```



Signalons le code très utilse suivant qui permet de parcourir tous les champs sur la base de leur nom:

```
Dim TB As TextBox

For Each TB In UserForm01.MultiPage2.Pages(2)
   MsgBox "Voilà la valeur dans le champ " & TB.Name & ": " & TB.Value
Next
```

Voyons ici un code très important qui permet d'associer de récupérer via une routine et une classe la valeur d'une famille de champs de textes (cela permet d'étiver d'écrire des codes identiques à rallonge). Pour cela, il faudra d'abord mettre **dans le code du formulaire** le code suivant:

```
Private m_colTBoxes As Collection

Private Sub UserForm_Initialize()
    Dim lngIndex As Long
    Dim intNumberFields As byte
    Dim clsTBox As Class1

    'on mettra ici le nombre de champs de saisie se trouvant sur le form
    intNumberFields=2

    Set m_colTBoxes = New Collection
    For lngIndex = 1 To 2
        Set clsTBox = New Class1
        'remarquez que ce code pour fonctionner suppose que les champs ont
        'tous un nom du type "Textbox1", "Textbox2",....
        Set clsTBox.TBox = Me.Controls("Textbox" & lngIndex)
        m_colTBoxes.Add clsTBox, CStr(m_colTBoxes.Count + 1)
    Next
End Sub
```

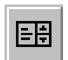

Ensuite, dans un module de classe qui s'appellera **Class1**, on met le code suivant:

```
Public WithEvents TBox As MSForms.TextBox

Private Sub TBox_Change()
    MsgBox "Validate " & Me.TBox.Name
End Sub
```

Ensuite il n'y a plus qu'à laisser marcher son imagination une fois cette structure de base connue!

15.4.4 ListBox ou zone de liste

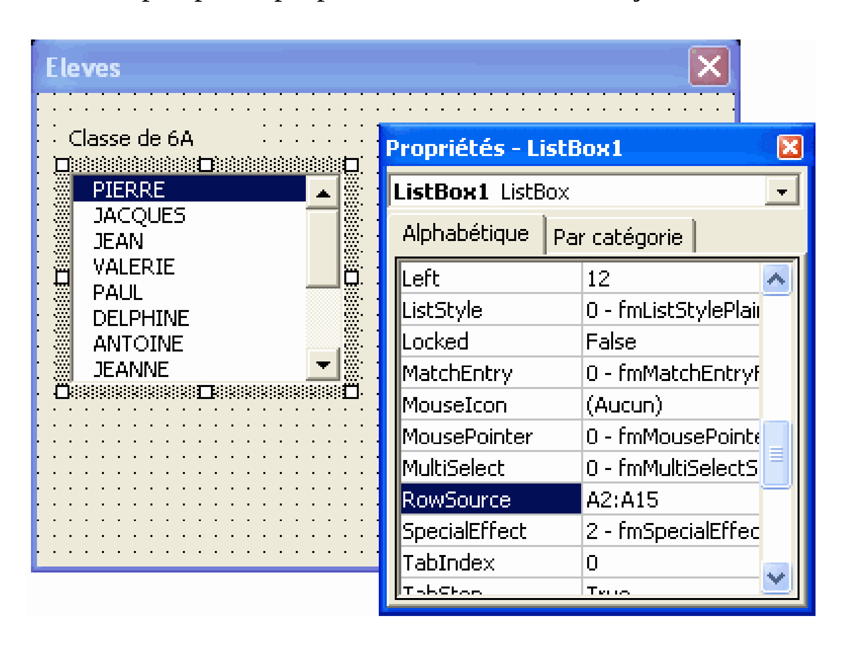
 Une zone de liste permet d'afficher une liste d'éléments sélectionnables par l'utilisateur.

Remarque: Attention les listbox on parfois des problèmes de hauteurs. Il peut arriver que l'on ne voie pas en entier le dernier élément à sélectionner.

Reprenons la liste d'élèves:

	A	B
1	ELEVE	NOTE
2	PIERRE	5
3	JACQUES	15
4	JEAN	10
5	VALERIE	12
6	PAUL	18
7	DELPHINE	13
8	ANTOINE	0
9	JEANNE	6
10	ANDRE	19
11	JOCELYNE	8
12	FELIX	12
13	JUSTIN	15
14	ETIENNE	6
15	MARIE	5

Les listes peuvent de remplir par la propriété `RowSource` de l'objet `ListBox`.



La méthode `AddItem` d'un objet ListBox permet d'ajouter un élément à la liste. La syntaxe est `ListBox.AddItem` "Texte", Index (Index correspond à l'emplacement du nouvel élément dans la liste). L'index du premier élément d'une liste a pour valeur 0. Si l'index n'est pas indiqué, le nouvel élément sera placé à la fin de la liste.

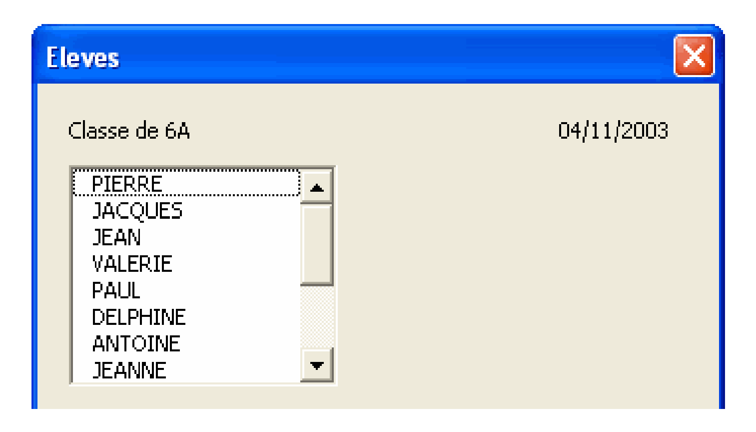
```
'La liste peut se remplir par la procédure suivante
Private Sub UserForm_Initialize()
    'Appel de la procédure Liste située dans un module
    Call Liste
End Sub

Sub Liste()

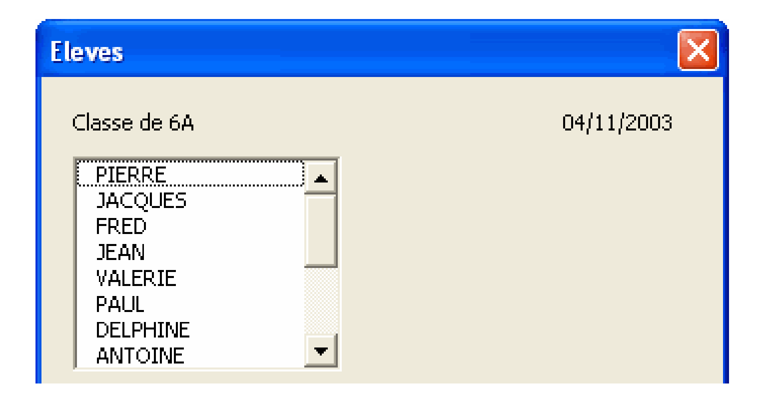
    Dim i As Integer

    For i = 1 To 14
        MaBoite.ListBox1.AddItem Range("A1").Offset(i)
    Next i
```

```
End Sub
```




```
'Ajout d'un élément situé en 2ème position
 MaBoite.AddItem "FRED", 2
```



La propriété `ListCount` d'une zone de liste renvoie le nombre d'éléments qu'elle contient, la propriété Value sa valeur et la propriété `ListIndex` son index.



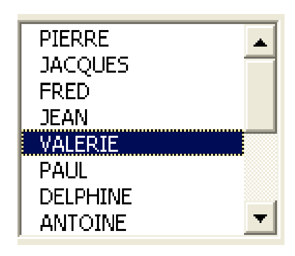

```
Private Sub ListBox1_Change()
    Dim i, j As Integer
    Dim Val As String
    i = ListBox1.ListCount 'renvoie 15
    j = ListBox1.ListIndex  'renvoie 4
    msgbox ListBox1.List(4) 'afficher "VALERIE"
    Val = ListBox1.Value    'renvoie "VALERIE"
End Sub
```

Il est également possible de remplir une zone de liste en utilisant un tableau et la propriété List.

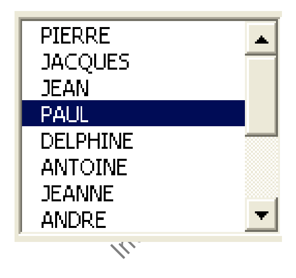
```
Sub Liste()
    Dim i As Integer
```

```
    Dim List_Eleve(1 To 14) As String
    For i = 1 To 14
        List_Eleve(i) = Range("A1").Offset(i)
    Next i
    MaBoite.ListBox1.List = List_Eleve
End Sub
```

La suppression d'un élément d'une liste se fait par la méthode RemoveItem. La syntaxe est ListBox.RemoveItem Index. Index correspond à l'élément à supprimer.

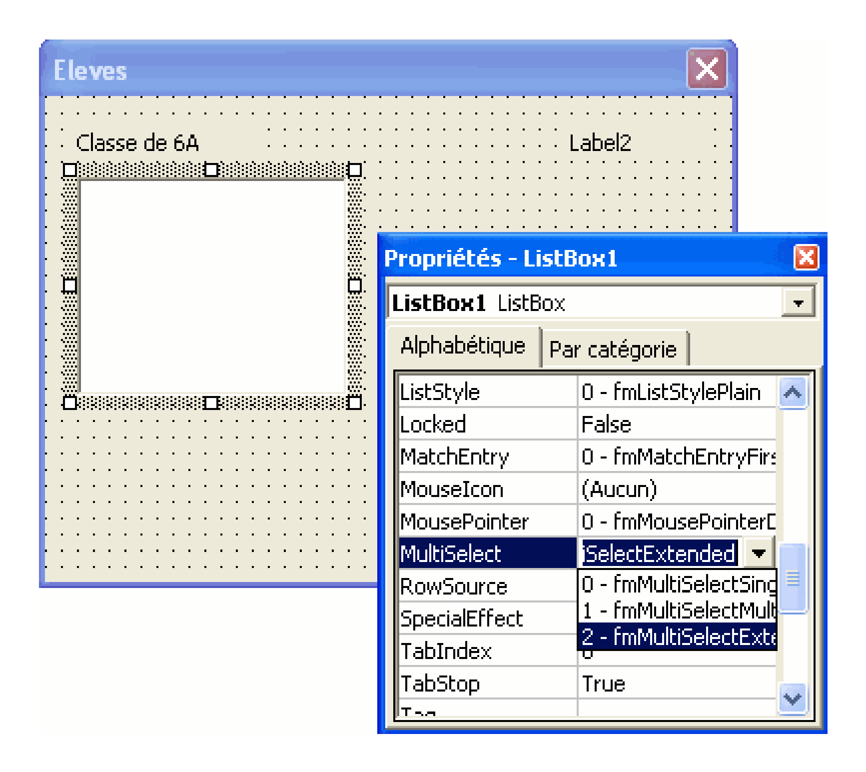



```
Dim i As Integer
i = ListBox1.ListIndex  'renvoie 4
ListBox1.RemoveItem i
```



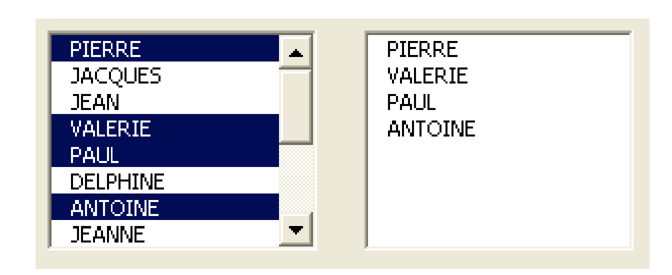
La suppression de tous les éléments d'une liste se fait par la méthode `Clear`. La syntaxe est `ListBox.Clear`.

Par défaut, l'utilisateur ne peut sélectionner qu'un seul élément de la liste. Pour permettre la sélection de plusieurs éléments, la propriété `MultiSelect` de la zone de texte doit être sur 1 ou sur 2.

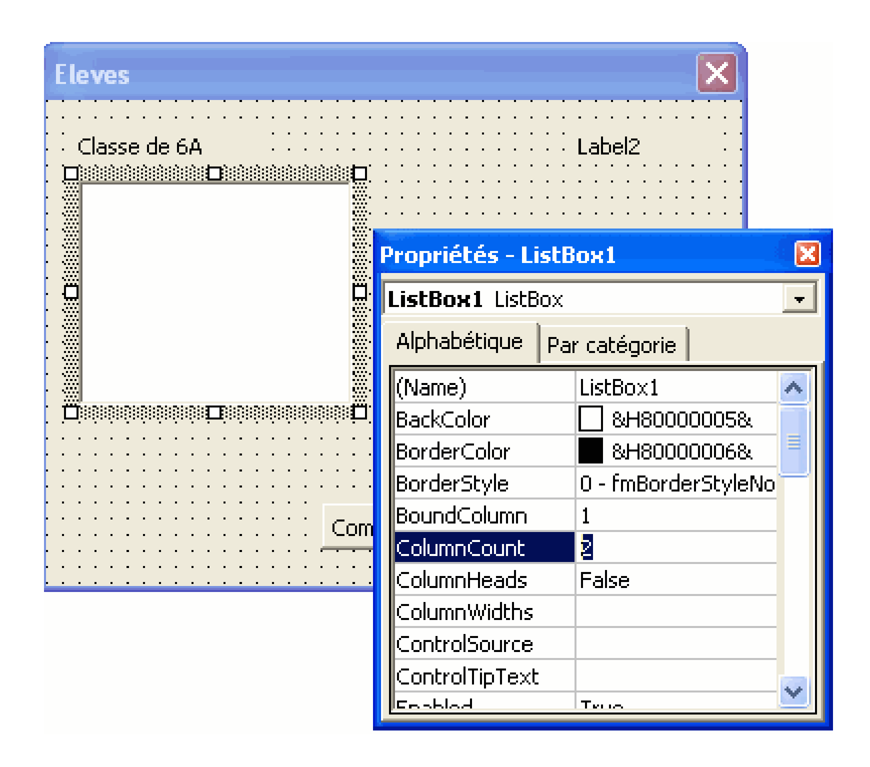


La propriété `Selected(Item)` détermine si un élément est sélectionné ou non. L'exemple suivant va copier les éléments sélectionnés de la ListBox1 dans la ListBox2.

```
Dim i As Integer
For i = 0 To ListBox1.ListCount - 1
    If ListBox1.Selected(i) = True Then
        ListBox2.AddItem ListBox1.List(i)
    End If
Next i
```

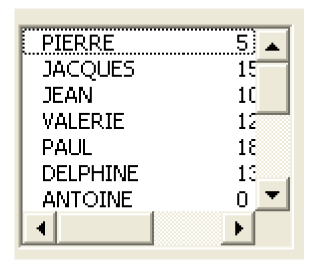
Une zone de liste peut contenir plusieurs colonnes. Le nombre de colonnes est défini par la propriété ColumnCount. Dans l'exemple suivant, la zone de liste va être composée de deux colonnes.



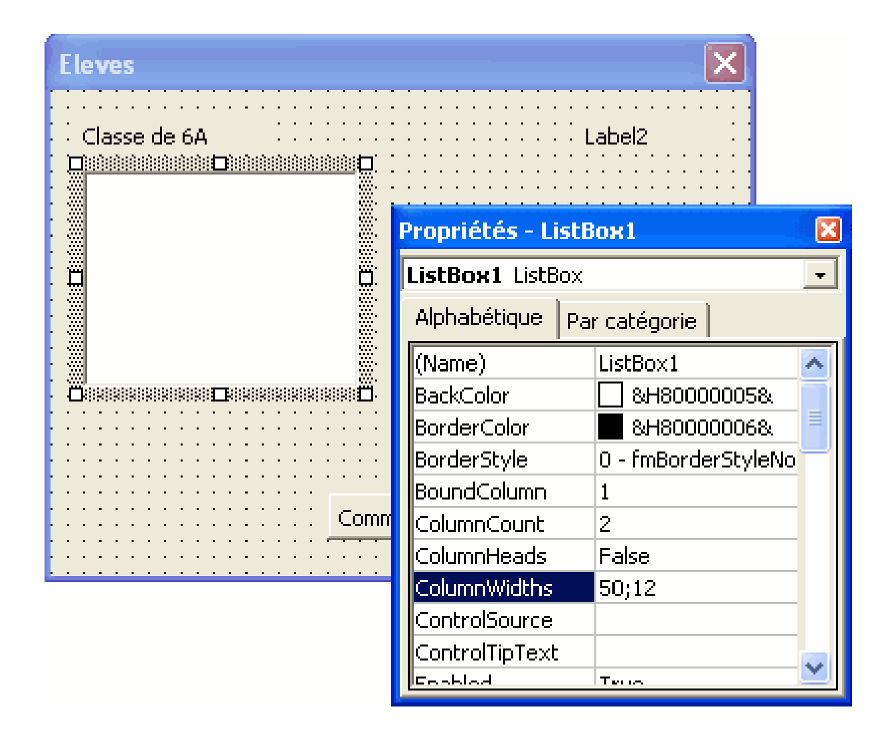
Pour remplir une zone de liste avec plusieurs colonnes, on va utiliser un tableau à plusieurs dimensions.

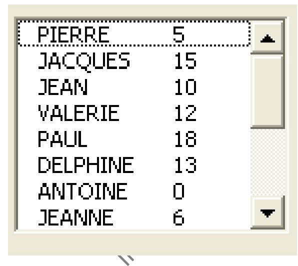
Dans l'exemple suivant, la zone de liste va recevoir le nom des élèves avec leurs notes.


```
Sub Liste()
    Dim i As Integer
    'On est obligé de passes par un tableau au préalable à ma connaissance…
    Dim List_Eleve(1 To 14, 1 To 2) As String 'dans la paranthèse il s'agit
du nombre de lignes, le nombre de colonnes
    For i = 1 To 14
        List_Eleve(i, 1) = Range("A1").Offset(i)
        List_Eleve(i, 2) = Range("B1").Offset(i)
    Next i
    MaBoite.ListBox1.List = List_Eleve
End Sub
```



La largeur de chaque colonne d'une zone de liste se change par la propriété `ColumnWidths`. Les différentes largeurs sont séparées par le caractère ";".



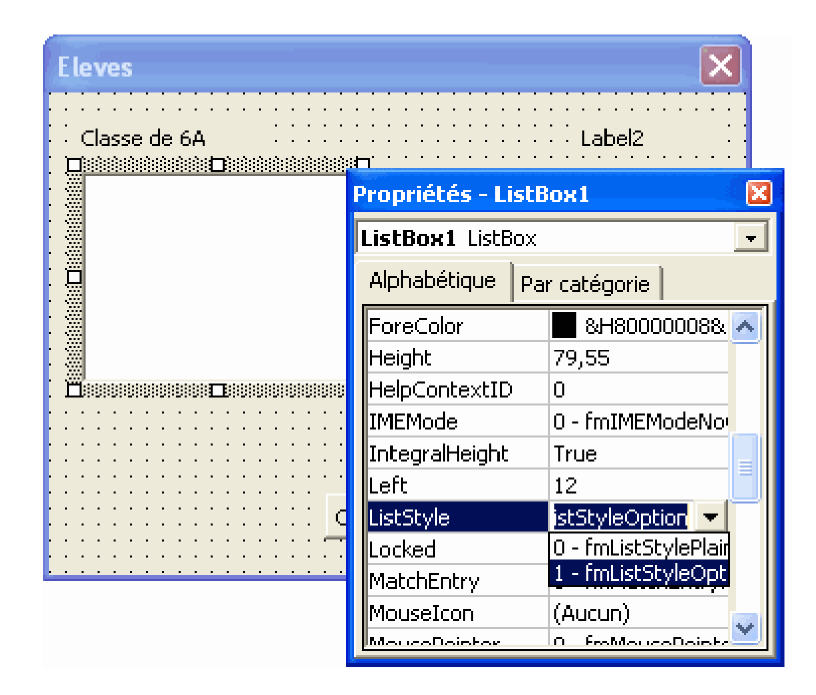


La propriété `BoudColumn` détermine dans quelle colonne la valeur est récupérée.

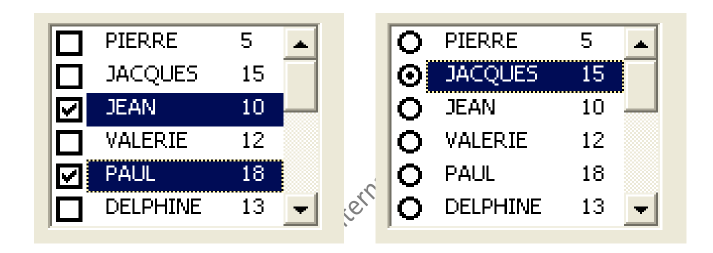



```
Dim Val As String
ListBox1.BoundColumn = 1
Val = ListBox1.Value     'renvoie "VALERIE"
ListBox1.BoundColumn = 2
Val = ListBox1.Value     'renvoie "12"
```

Il est possible de changer l'aspect d'une zone de liste par la propriété `ListStyle`.



L'aspect de la zone de liste change selon la valeur de la propriété `MultiSelect`.



Pour choper la valeur d'une sélection d'une listbox à choix multiples, il faut utiliser la propriété `.ListIndex` qui donne le numéro de la ligne sélectionnée. Ensuite pour prendre la valeur correspondante à une colonne X donnée il faut faire:

```
Sub ListBox1_Change()
   Arr_List =  ListBox1.List
   MsgBox Arr_List(ListBox1.ListIndex,X)
End Sub
```

Pour cocher un élément en V.B.A. (par exemple le quatrième) il suffit d'utiliser `ListBox1.Selected(3) = True`

Remarque: La plupart des utilisateurs veulent trier les éléments dans les listbox (ou combobox). Il faut coder pour cela. Voyons un exemple avec notre List_Box1:

```
For j=0 to List_Box1.ListCount
    For i=1 to List_Box1.ListCount-1
      'attention à mettre ucase (ou) lcase lorsque notre
      If List_Box1.List(i-1)>List_Box1.List(i) then
            strTemp=List_Box1.List(i-1)
            List_Box1.List(i-1)=List_Box1.List(i)
            List_Box1.List(i)=strTemp
      End If
    Next i
Next j
```

15.4.5 ComboBox ou liste déroulante

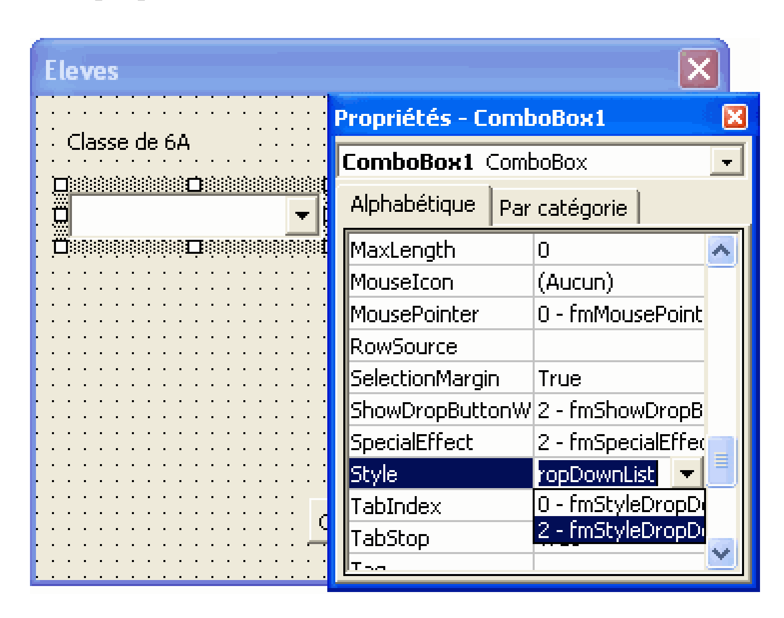
 Une liste déroulante se remplit de la même façon qu'une zone de liste.

Contrairement à la zone de liste, la liste déroulante peut permettre à l'utilisateur de saisir une chaîne de caractères qui ne fait pas partie de la liste.



Si la valeur saisie ne fait pas partie de la liste ou est nulle, la propriété `ListIndex` de l'objet ComBox prend comme valeur -1.

Il est possible d'interdire à l'utilisateur de saisir une chaîne de caractère qui ne fait pas partie de la liste en mettant la propriété `Style` sur 2.

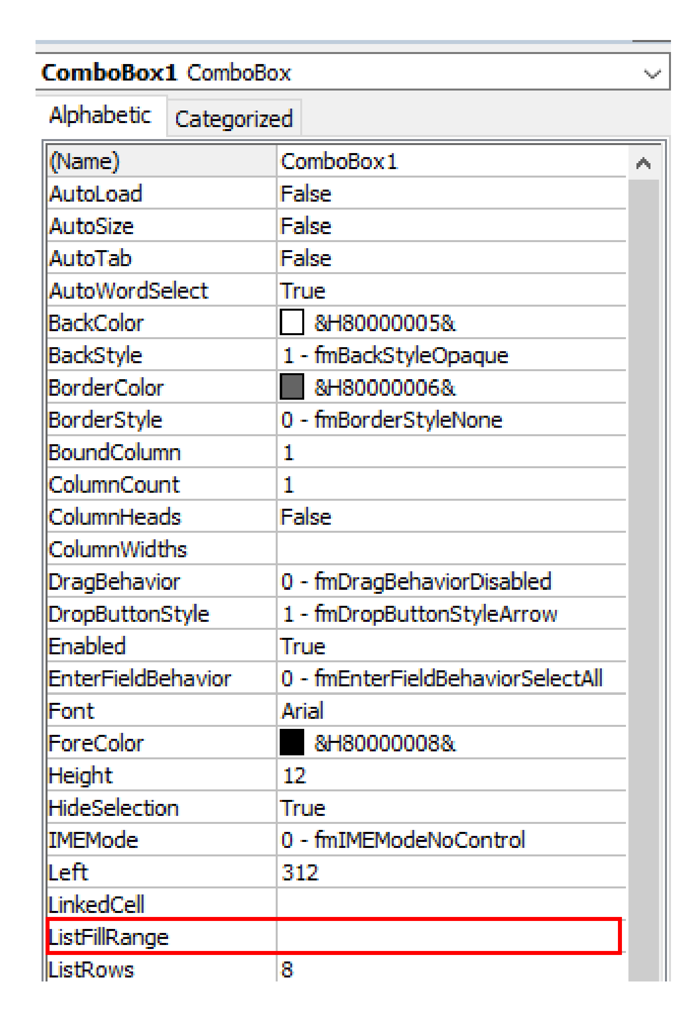


Attention!!! Si vous insérez un ActiveX de type ComboBox dans une feuille (sheet) Excel il faut éviter de la peupler sur un événement qui est propre à elle-même (vous aurez des problèmes!). Utilisez si possible toujours l'événement de feuille Worksheet_Activate

Une manière très courante de remplir un combo box c'est lors de l'activation du UserForm sur laquelle elle se trouve. Par exemple en utilisant un range d'une feuille qui serait cachée (choix qui simplifie le code grandement mais qui à ses désavantages...):

```
Private Sub UserForm_Initialize()
    Me.ComboBox1.List = Worksheets("Sheet1").Range("B12:B376").Value
End Sub
```

Notez que les propriétés des combobox diffèrent en fonction de si vous les créez dans un userform ou dans une feuille Microsoft Excel. Par exemple, pour remplir le contenu nous avons cette propriété *ListFillRange* lorsque le contrôle est créé sur une feuille mais qui n'apparaît pas lorsqu'il est créé dans un userform:

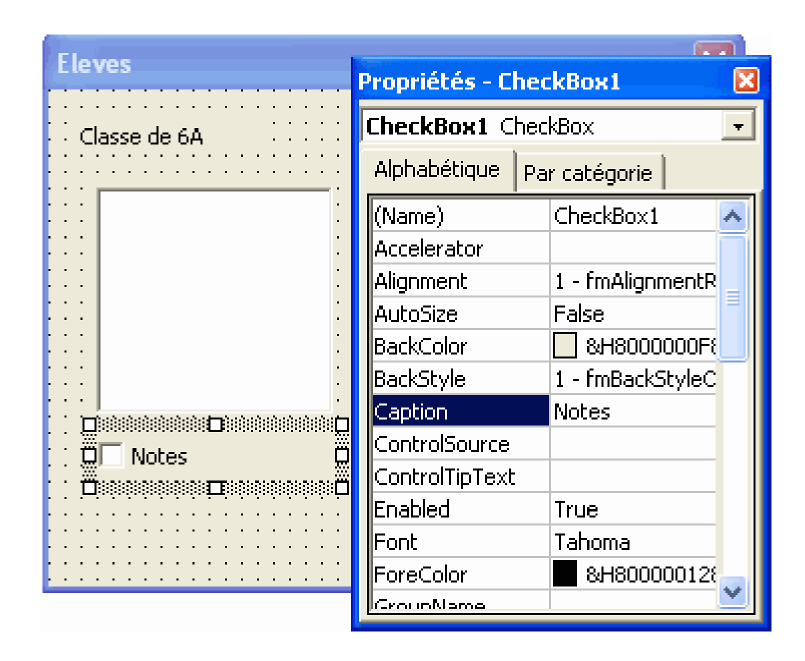


Or to add for example the list of all Excel sheets:

```
Private Sub UserForm_Initialize()
    Dim I As Long
    Me.ComboBox1.Clear
    For I = 1 To Sheets.Count
        Me.ComboBox1.AddItem Sheets(I).Name
    Next
    Me.ComboBox1.Value = ActiveSheet.Name
End Sub
```

15.4.6 CheckBox ou case à cocher

Cet outil crée des cases que l'utilisateur peut activer ou désactiver d'un simple click



Si la case à cocher est activée, sa propriété `Value` prend comme valeur `True`, sinon elle prend comme valeur `False`.

Dans l'exemple suivant, si la case à cocher est activée, les notes apparaissent, sinon elles disparaissent.

```
Private Sub UserForm_Initialize()

    Dim i As Integer
    Dim List_Eleve(1 To 14, 1 To 2) As String
    For i = 1 To 14
        List_Eleve(i, 1) = Range("A1").Offset(i)
        List_Eleve(i, 2) = Range("B1").Offset(i)
    Next i
    MaBoite.ListBox1.List = List_Eleve

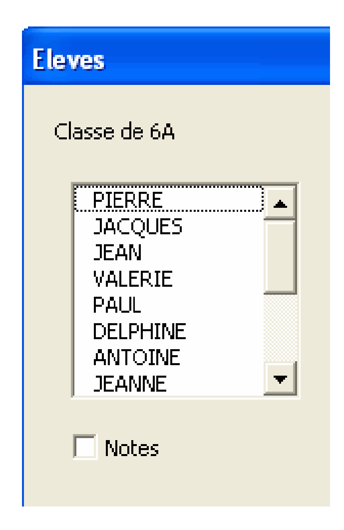
End Sub
```

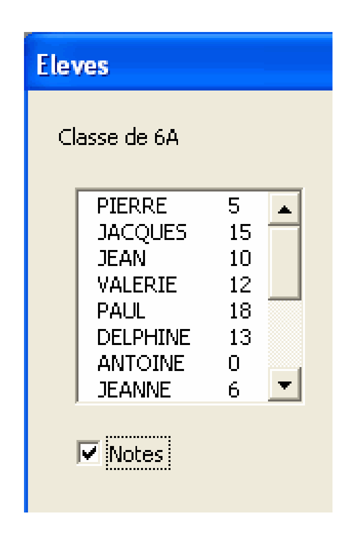


```
Private Sub CheckBox1_Click()

    If CheckBox1.Value = True Then
        ListBox1.ColumnCount = 2
    Else
        ListBox1.ColumnCount = 1
    End If

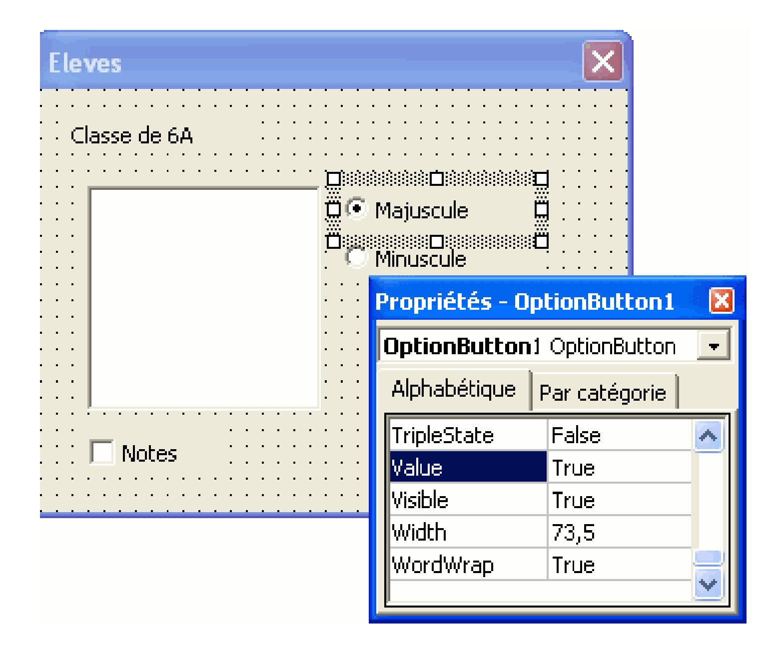
End Sub
```





15.4.7 OptionButton ou bouton d'option

Cet outil crée des boutons de d'options. L'utilisateur ne peut sélectionner qu'un seul bouton d'un même groupe.



La propriété Value du bouton sélectionné prend la valeur True alors que la propriété Value des autres boutons du même groupe prend la valeur False.

```
Private Sub OptionButton1_Click()

    Ecrire(True)

End Sub

Private Sub OptionButton2_Click()

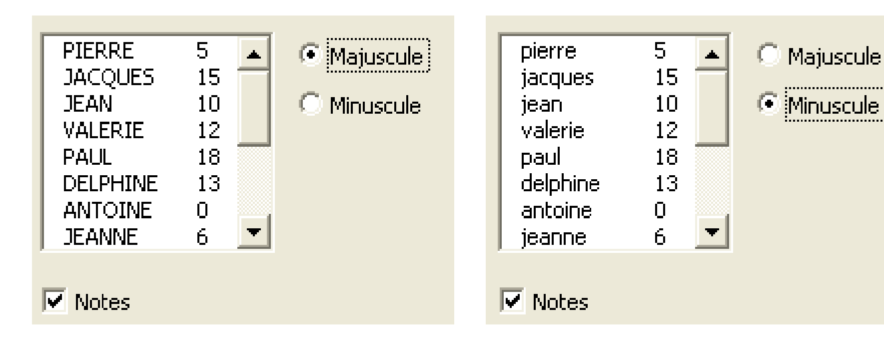
    Ecrire(False)

End Sub
```

```
Sub Ecrire(Maj As Boolean)

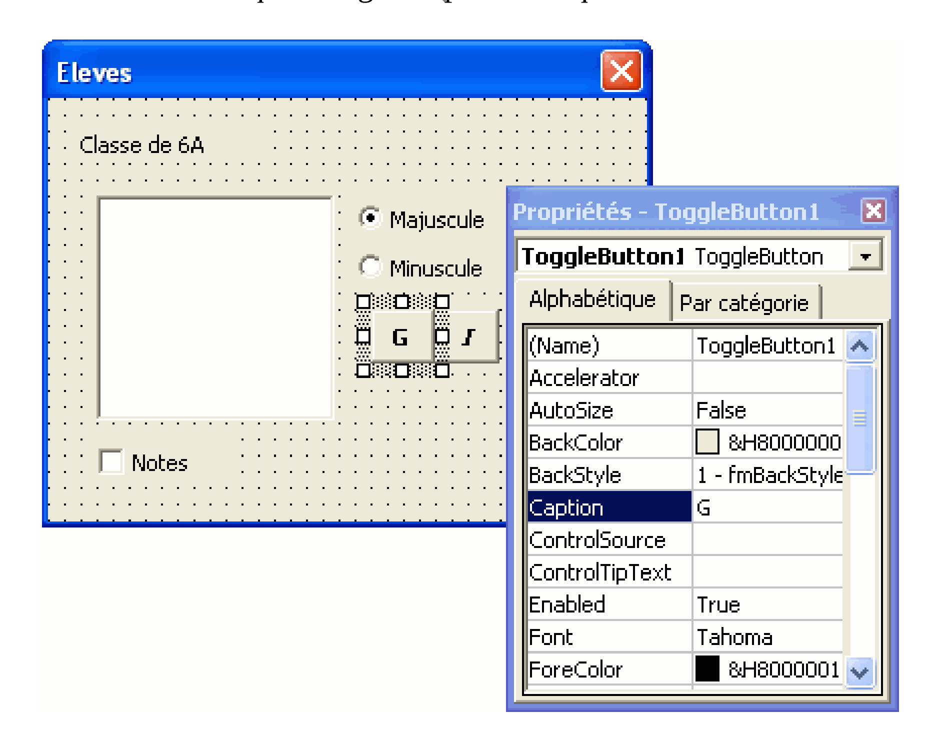
    Dim i As Integer
    For i = 0 To ListBox1.ListCount - 1
        If Maj = True Then
            'Majuscule
            ListBox1.List(i) = UCase(ListBox1.List(i))
        Else
            'Majuscule
            ListBox1.List(i) = LCase(ListBox1.List(i))
        End If
    Next i

End Sub
```



15.4.8 ToggleButton ou Bouton à bascule

Cet outil crée un bouton qui change d'aspect à chaque click.

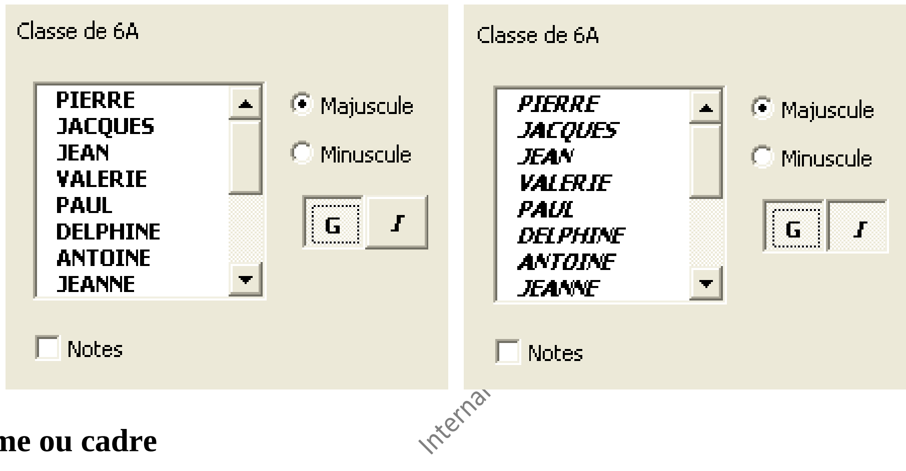


Si le bouton est enfoncé, sa valeur est égale à `True` sinon elle est égale à `False`.

```
'Bouton Gras
Private Sub ToggleButton1_Click()
```

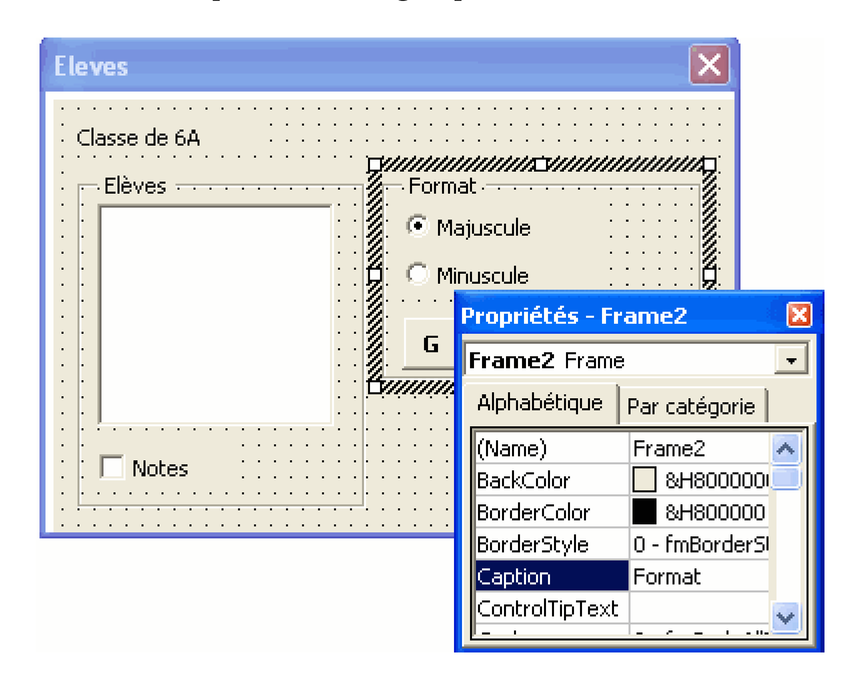
```
    If ToggleButton1 = True Then
        ListBox1.Font.Bold = True
    Else
        ListBox1.Font.Bold = False
    End If
End Sub
'Bouton Italic
Private Sub ToggleButton2_Click()
    If ToggleButton2 = True Then
        ListBox1.Font.Italic = True
    Else
        ListBox1.Font.Italic = False
    End If

End Sub
```

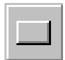


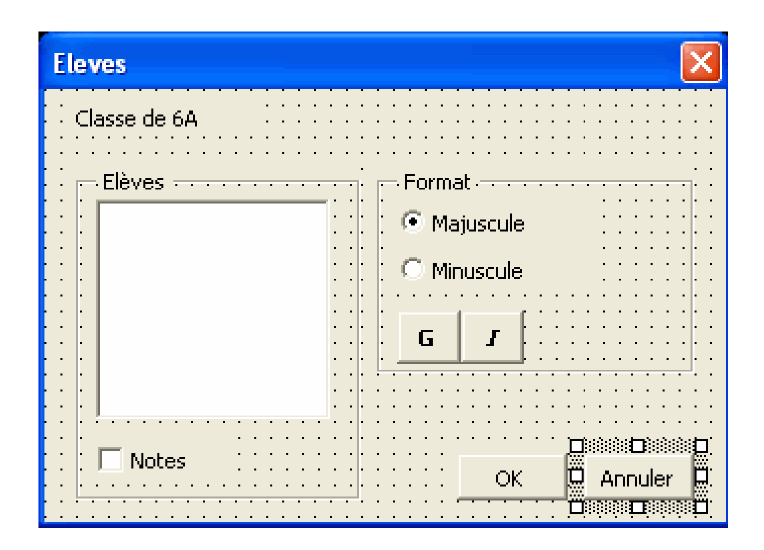
15.4.9 Frame ou cadre

Cet outil crée des cadres permettant de grouper des contrôles.

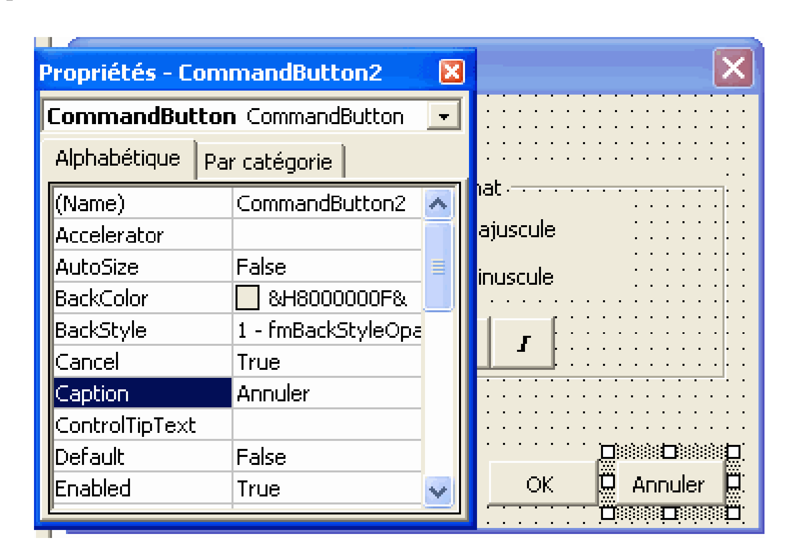


15.4.10 CommandButton ou Bouton de commande

 Cet outil crée des boutons de commande tel que des boutons OK ou Annuler.



Si vous affectez la valeur `True` à la propriété `Default` d'un bouton de commande et si aucun autre contrôle n'est sélectionné, la touche **Entrée** équivaut à un clic sur ce même bouton. De même, si vous affectez la valeur `True` à la propriété `Cancel` d'un bouton de commande, la touche **Echap** équivaut à un clic sur le bouton.

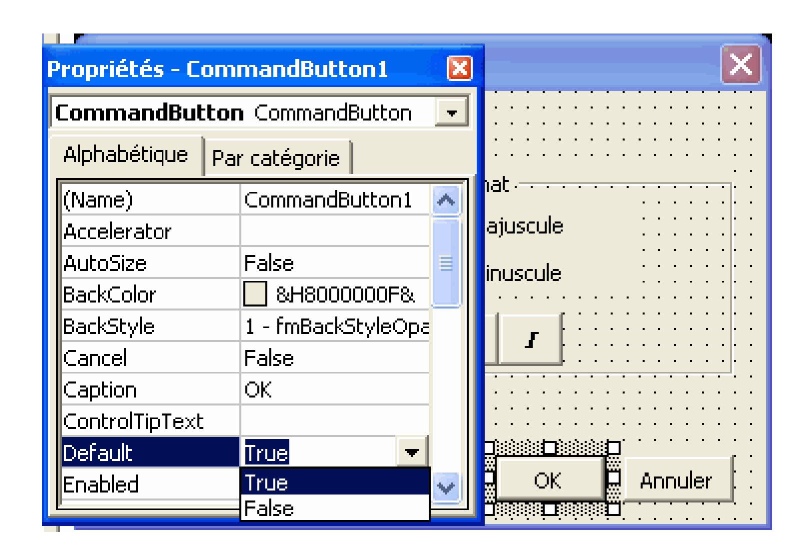


Dans cet exemple, le fait de taper sur la touche **Echap** équivaut à un clic sur le bouton **Annuler** et ferme le UserForm

```
'Bouton Annuler
Private Sub CommandButton2_Click()

    Unload MaBoite

End Sub
```



Dans cet exemple, le fait de taper sur la touche **Entrée** équivaut à un clic sur le bouton OK si aucun autre contrôle n'est sélectionné et met à jour la liste dans la feuille de calcul.

```
'Bouton OK
Private Sub CommandButton1_Click()

    MAJListe

End Sub

Sub MAJListe()

    Dim NreENreg as Integer
    NbreEnreg = ListBox1.ListCount
    Range("A2:B" & NbreEnreg + 1) = ListBox1.List

End Sub
```

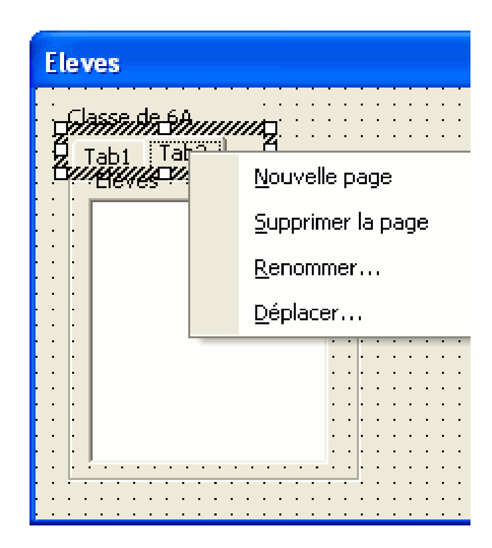

15.4.11 TabStrip ou étiquette d'onglet


Cet outil crée des étiquettes d'onglet pour des pages identiques.

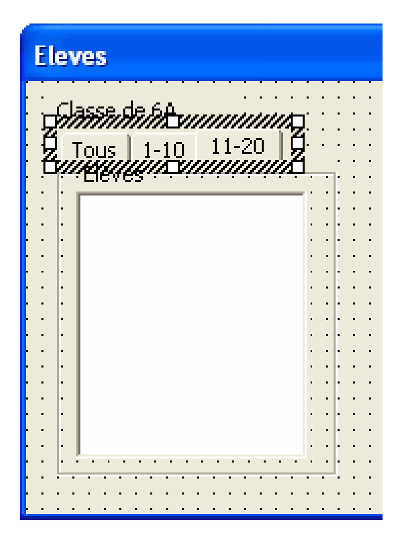
Par défaut, le nombre d'onglets d'un nouveau TabStrip est de 2 et sont nommés Tab1 et Tab2. Un simple clic avec le bouton droit de la souris permet d'en ajouter, de les renommer, de les déplacer ou de les supprimer.

Attention par défaut un contrôle créé sur un TabStrip sera créé sur tous les autres Tab car en réalité il faut gérer le changement de Tab en V.B.A. et masquer/afficher les contrôles en fonction du contexte. Raison pour laquelle on préférera souvent les onglets de type Pages (voir plus loin).

Le TabStrip est conseillé lorsque les objets sur chaque page sont identiques, alors que le MultiPage est conseillé lorsque les objets sur chaque page sont différents.



Dans l'exemple suivant, ajoutons un TabStrip avec 3 onglets permettant de classer les élèves suivant leurs notes.

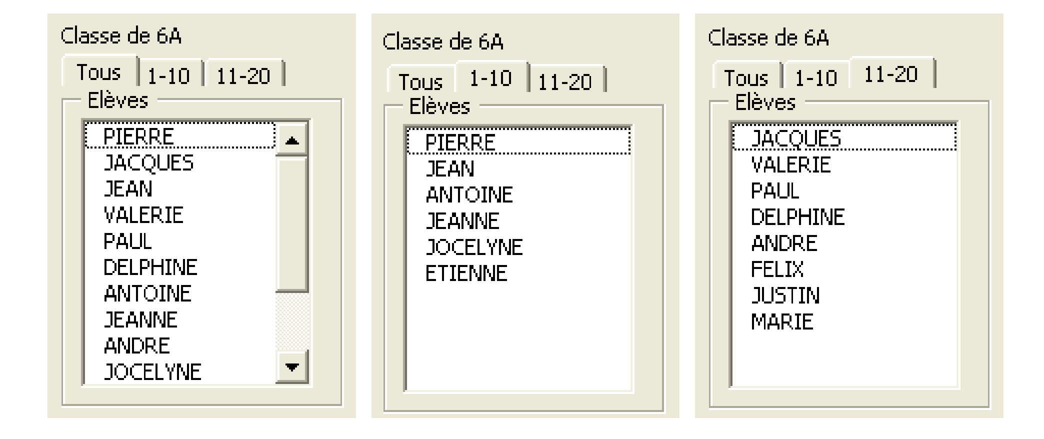


L'onglet sur lequel clique l'utilisateur est déterminé par la propriété Value du Tab Strip qui prend comme valeur 0 si l'utilisateur clique sur le premier onglet, 1 s'il clique sur le second, 3 si il clique sur le troisième ...

```
Private Sub TabStrip1_Click()
    Liste(TabStrip1.Value)
End Sub

Sub Liste(OptNote As Integer)
    Dim Note As Integer
    Dim Eleve As String
    Dim i As Integer
    Dim NoteMini As Integer
    Dim NoteMaxi As Integer
    MaBoite.ListBox1.Clear 'Efface le contenu de la liste
    Select Case OptNote
    Case 0 'Toutes les notes(onglet 1)
        NoteMini = 0
        NoteMaxi = 20
    Case 1 'Notes de 0 à 10(onglet2)
        NoteMini = 0
```

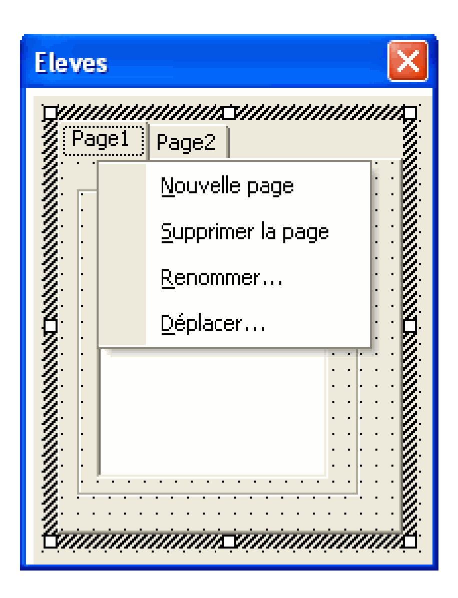
```
        NoteMaxi = 10
    Case 2 'Notes de 11 à 20(onglet3)
        NoteMini = 11
        NoteMaxi = 20
    End Select
    For i = 1 To 14
        Note = Range("B1").Offset(i)
        If Note >= NoteMini And Note <= NoteMaxi Then
            Eleve = Range("A1").Offset(i)
            MaBoite.ListBox1.AddItem Eleve
        End If
    Next i
End Sub
```



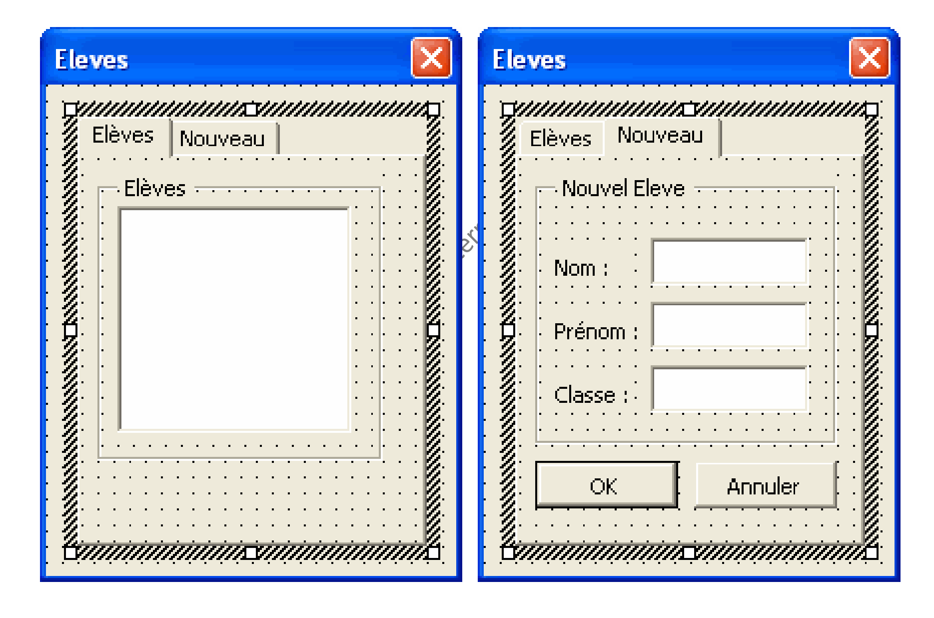
15.4.12 MultiPage

Un multipage peut être comparé à plusieurs UserForm dans le même. Tout comme le TabStrip, le multipage contient des onglets mais à chaque onglet correspond une nouvelle page qui contient des contrôles différents. Sa création est identique à la création d'un TabStrip.

La différence principale entre les Multipages et les TabStrip restent quand même leurs propriétés (images de fonds, barres de défilements, etc.) accessibles dans la fenêtre des propriétés.



L'onglet sur lequel clique l'utilisateur est déterminé par la propriété Index du multipage qui prend comme valeur 0 si l'utilisateur clique sur le premier onglet, 1 si il clique sur le second, 3 si il clique sur le troisième ...

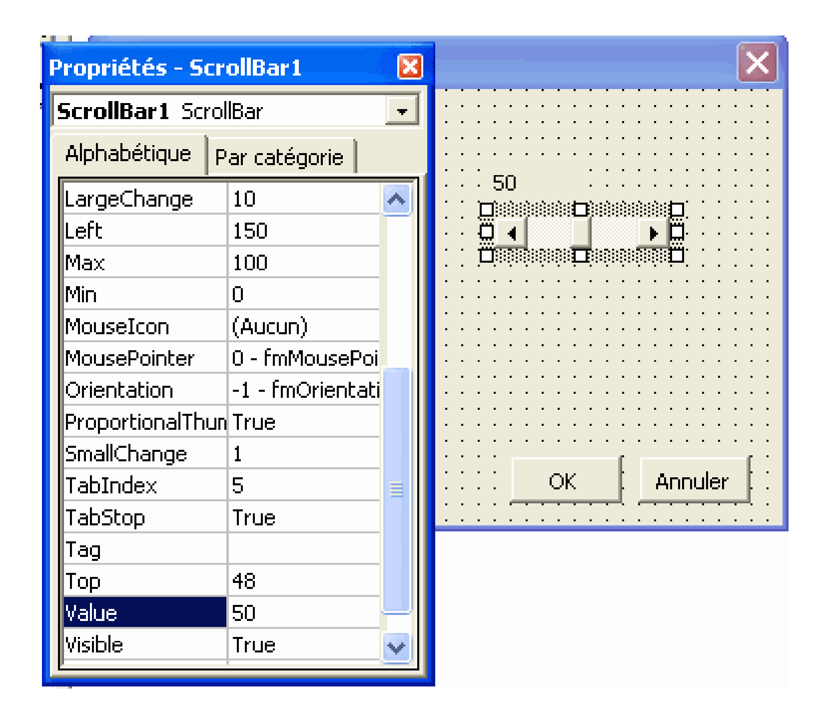


Indiquons un code important pour déterminer la page sur laquelle on se trouve et qui est très utile quand l'on gère des DatePicker (champ de calendrier):

```
MsgBox MultiPage1.SelectedItem.Caption
```

15.4.13 ScrollBar ou Barre de défilement

Une barre de défilement peut être horizontale ou vertical selon son redimensionnement. L'exemple suivant se compose d'une barre de défilement et d'une étiquette qui reçoit sa valeur.



La valeur mini d'une barre de défilement se définie par sa propriété `Min`, sa valeur maxi par sa propriété `Max` et sa valeur par sa propriété `Value`.

La propriété `LargeChange` définie le changement de valeur lorsque l'utilisateur clique entre le curseur et l'une des flèches.

La propriété `SmallChange` définie le changement de valeur lorsque l'utilisateur clique sur l'une des deux flèches.

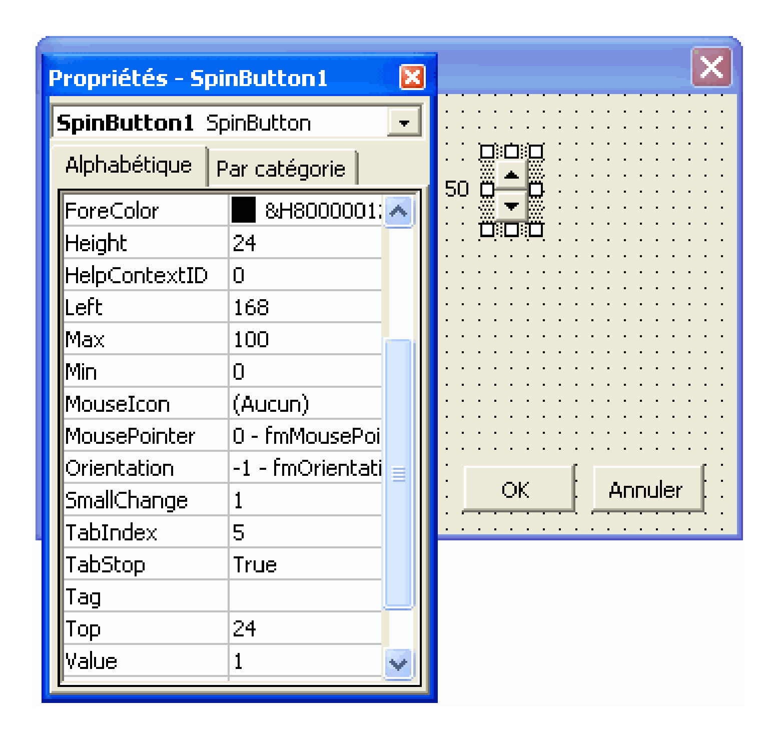
La propriété `Delay` définie le temps (en millisecondes) entre chaque changement lorsque l'utilisateur reste appuyer sur le Scrollbar.

L'étiquette prend la valeur de la barre de défilement par la procédure suivante:

```
Private Sub ScrollBar1_Click()
    Label1 = ScrollBar1.Value
End Sub
```

15.4.14 SpinButton ou Bouton rotatif

Le bouton rotatif possède presque les mêmes propriétés qu'une barre de défilement. Il ne peut cependant incrémenter ou décrémenter un nombre que de la même valeur (définie dans sa propriété `Value`) à chaque fois.

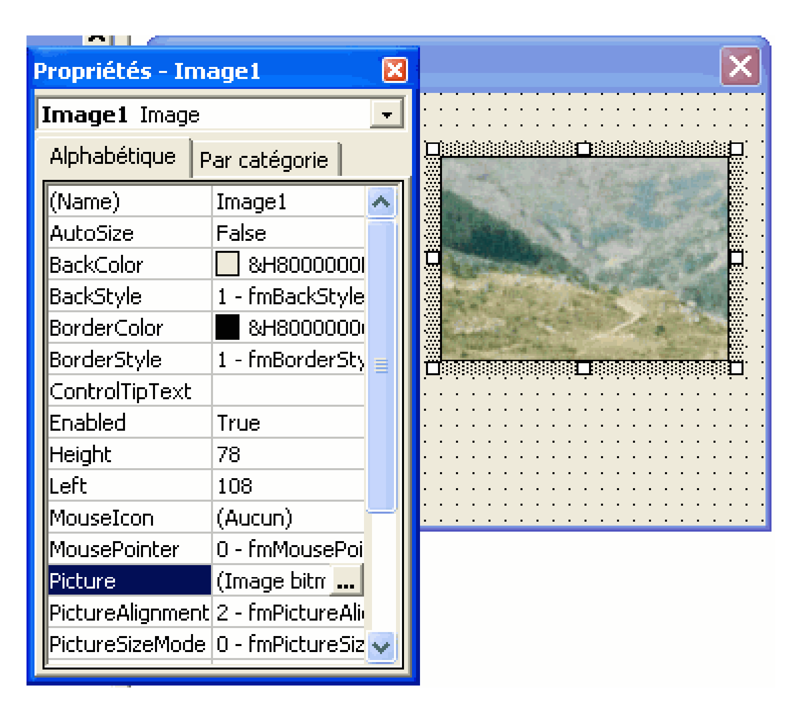



```
Private Sub SpinButton1_Click()
    Label1 = SpinButton1.Value
End Sub
```

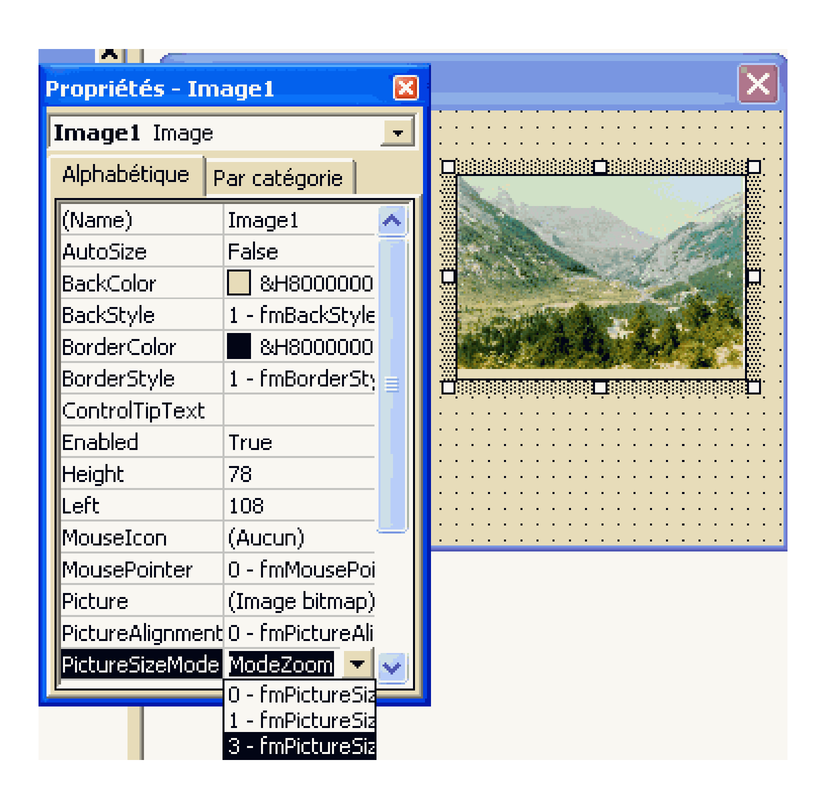
15.4.15 Image

Cet outil permet d'ajouter une image sur un UserForm.

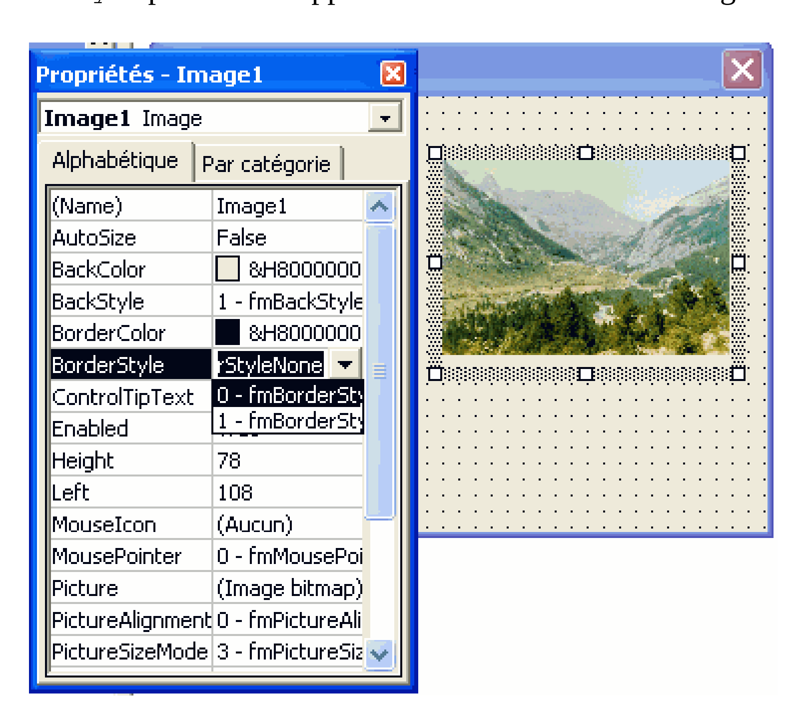
La sélection de l'image à placer se fait en cliquant sur ... de la propriété `Picture`:



La propriété `PictureSizeMode` permet de redimensionner l'image.



La propriété `BorderStyle` permet de supprimer le cadre autour de l'image.



Le code V.B.A. permet également de charger ou de décharger une image par la propriété Picture.

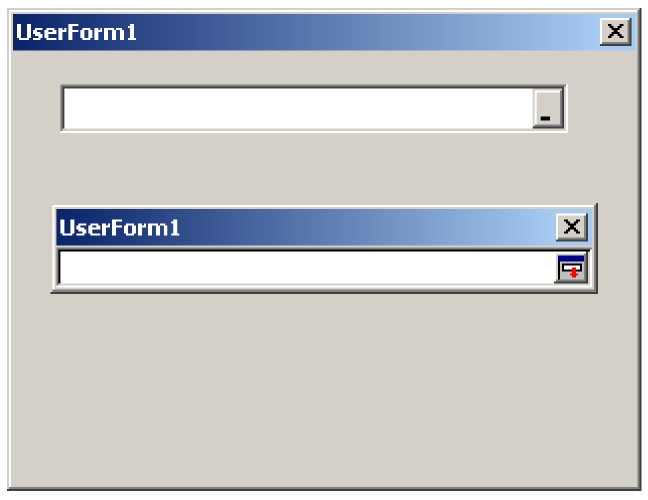
```
Dim Photo As String
Image1.Picture = LoadPicture() 'Décharge l'image
Photo = "c:\cheminphoto\photo.jpg"
Image1.Picture = LoadPicture(Photo) 'charge l'image
```

Le contrôle Image supporte les formats d'image bmp, cur, gif, ico, jpg, et wmf.

Évidemment à partir de là tout est possible: navigateur d'image, gestion de fichiers d'images, analyse d'images, etc.

15.4.16 RefEdit ou Éditeur de Référence

L'éditeur de référence permet d'ajouter dans un UserForm un élément de sélection de cellules:

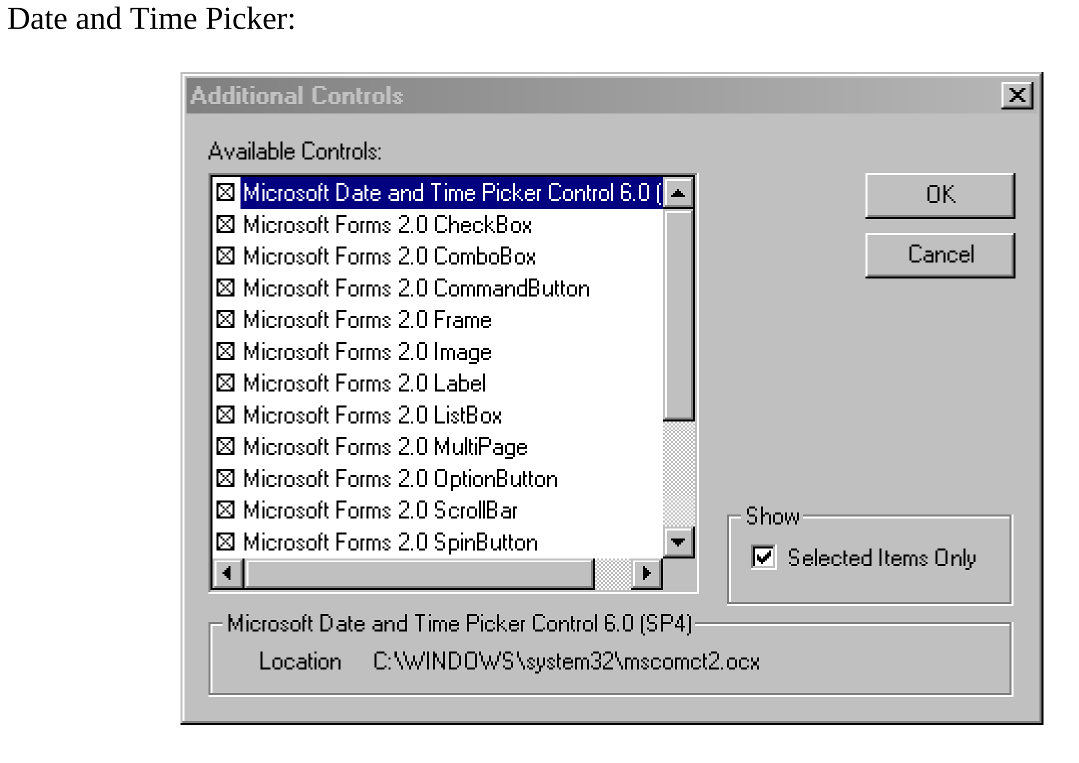


Son contenu (la sélection fait par l'utilisateur) peut être simplement obtenue par la commande:

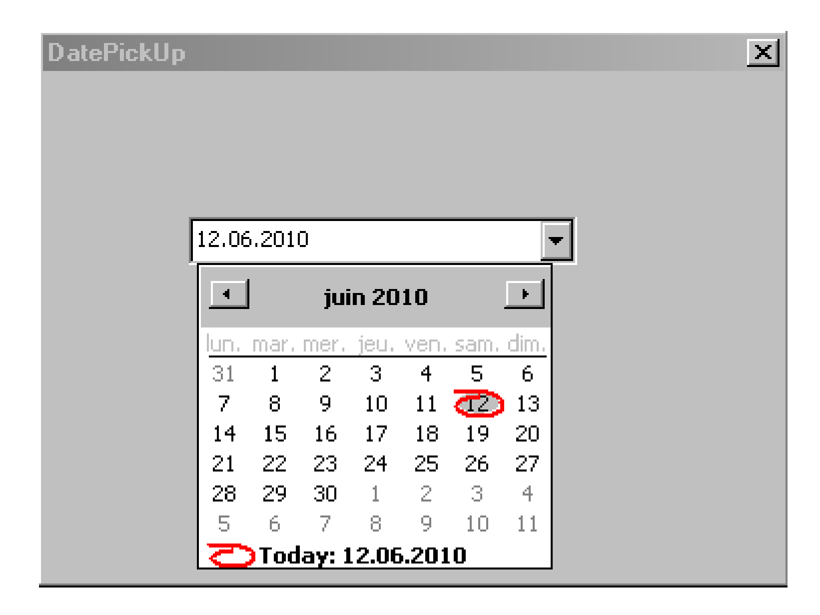
```
RefEdit1.Value
```


15.4.17 Contrôle Date PickUp

Il s'agit d'un simple champ permettant de sélectionner une date et qui se nomme Microsoft Date and Time Picker:

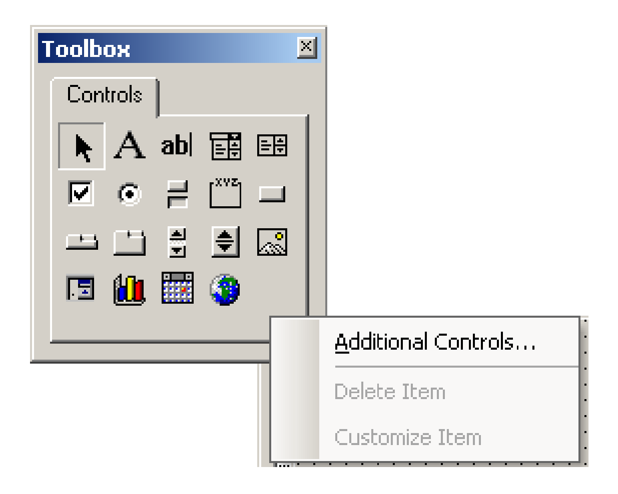


et qui ressemble à l'objet suivant une fois sur le formulaire:



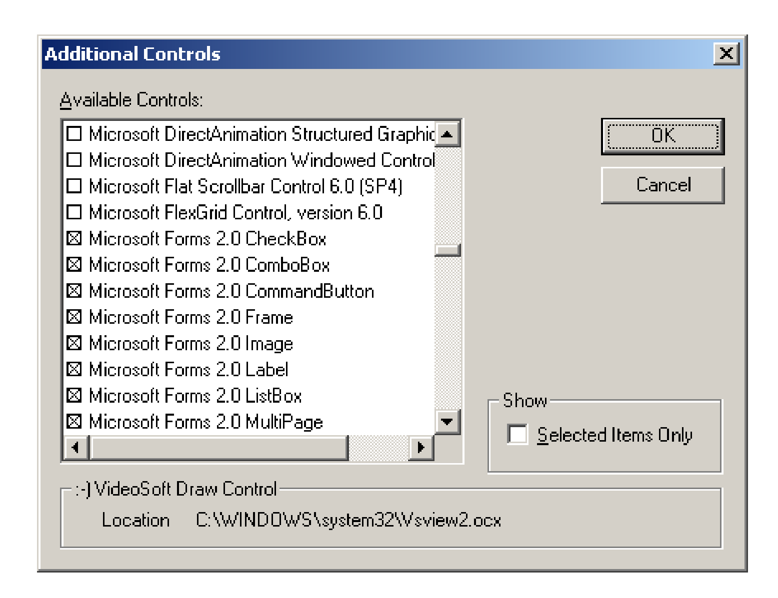
15.4.18 Contrôle Calendrier

Il est possible d'ajouter une centaine de contrôles supplémentaires dans la barre d'outils de contrôles. Pour cela il suffit de faire un clic droit sur cette barre:



et de sélection *Additional Controls*…

Apparaît alors la boîte suivante (remarquez la case à cocher dans le coin qui est parfois très utile!):

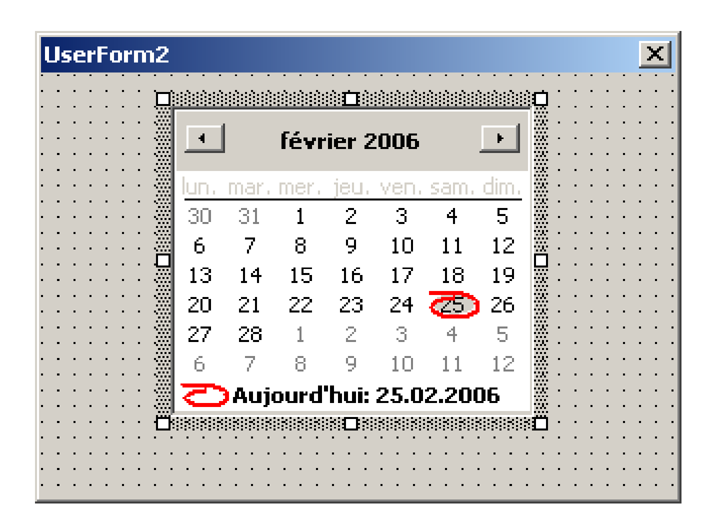


Dans la liste, nous allons nous intéresser au contrôle nommé:

Microsoft Month View Calendar

Une fois cochée et validé on voit s'ajouter dans *Toolbox* le bouton suivant:

Vous pouvez alors ajouter ce contrôle MonthView1 dans un nouveau userform tel que ci-dessous:



et y lire la valeur sélectionnée avec l'événement `DateClick` et la valeur avec la méthode `Value`:

```
Private Sub MonthView1_DateClick(ByVal DateClicked As Date)
     MsgBox MonthView1.Value
End Sub

Private Sub UserForm_Activate()
     MonthView1.Value=Now
End Sub
```

Avec Office 2007 et 2010, ce contrôle n'est plus disponible dans la liste des contrôles supplémentaires. Il faut rechercher le composant MSCOMCT2.ocx qui permet de l'utiliser.

Vous pouvez le télécharger à cette adresse:

http://support.microsoft.com/kb/297381/fr

Il suffira de décompresser le fichier *.cab dans le dossier Windows/System32.

Pour qu'il soit ensuite disponible dans V.B.A. il faudra l'inscrire dans la table de registre.

Dans le CMD tapez:

regsvr32 c:\windows\system32\mscomct2.ocx

et confirmez l'ajout. Il vous sela alors possible d'aller le rajouter à la boîte à outils sous le nom:

Microsoft Date and Time Picker Control 6.0 (SP4)

15.4.19 Contrôle Browser (navigateur)

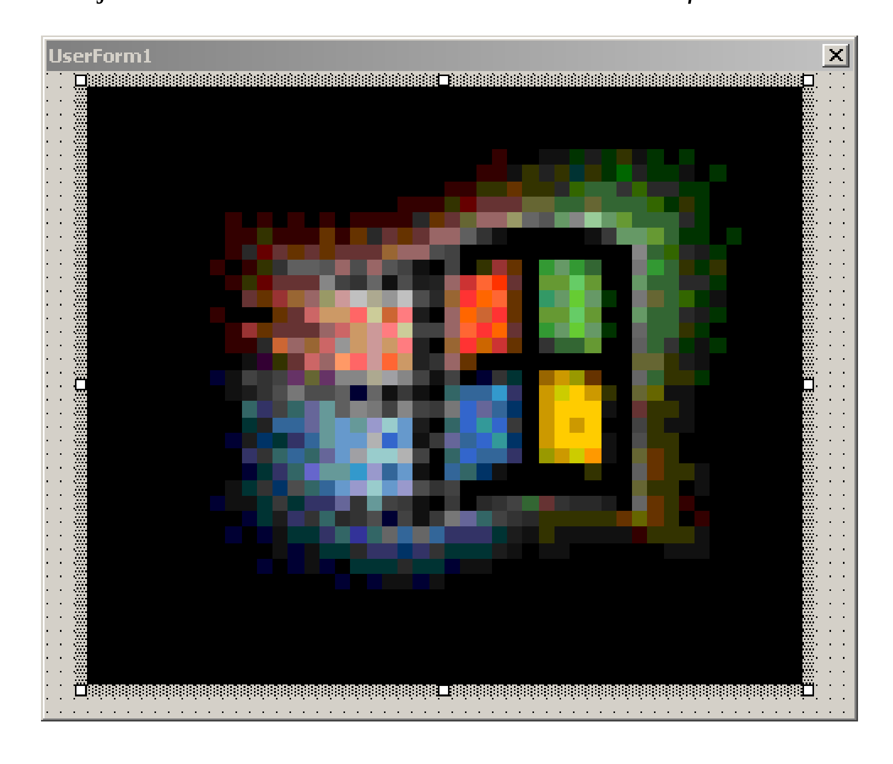
Nous allons nous intéresser maintenant à ajouter une fenêtre Internet Explorer dans un *Userform*. Nous pourrions par ailleurs aussi insérer ce contrôle directement sur une feuille Excel ou dans une diapositive PowerPoint.

Comme pour le calendrier, il faut ajouter ce contrôle qui se nomme:

Microsoft Web Browser

et qui ajoutera le bouton suivant dans la *Toolbox*:

Ensuite, il suffit d'ajouter ce contrôle *WebBrowser1* dans un *Userform*:



Ensuite, nous pouvons ajouter le code suivant sur l'initialisation du formulaire:

```
Private Sub UserForm_Initialize()
    WebBrowser1.Navigate http://www.microsoft.com
End Sub
```

Voyons un autre code utile qui permet d'identifier le navigateur IE ouvert en arrière-plan d'Excel et d'absorver le texte de la page dans une variable (pas besoin d'ajouter de références!!!)

```
Sub ReadPageContent()
   Dim ie As Object
   Dim objShell as Object
   Dim objWindow as Object
   Dim objItem as Object

   Set objShell = CreateObject("Shell.Application")
   Set objWindow = objShell.Windows()

   For Each objItem In objWindow

   Next objItem
End Sub
```

15.4.20 Contrôle Chart

Nous allons ici nous intéresser à l'insertion d'un graphique Excel simple dans un *Userform* ce qui peut être très utils dans des logiciels comme Excel, Access ou particulièrement Project.

Pour voir un exemple il faut d'abord ajouter le contrôle:

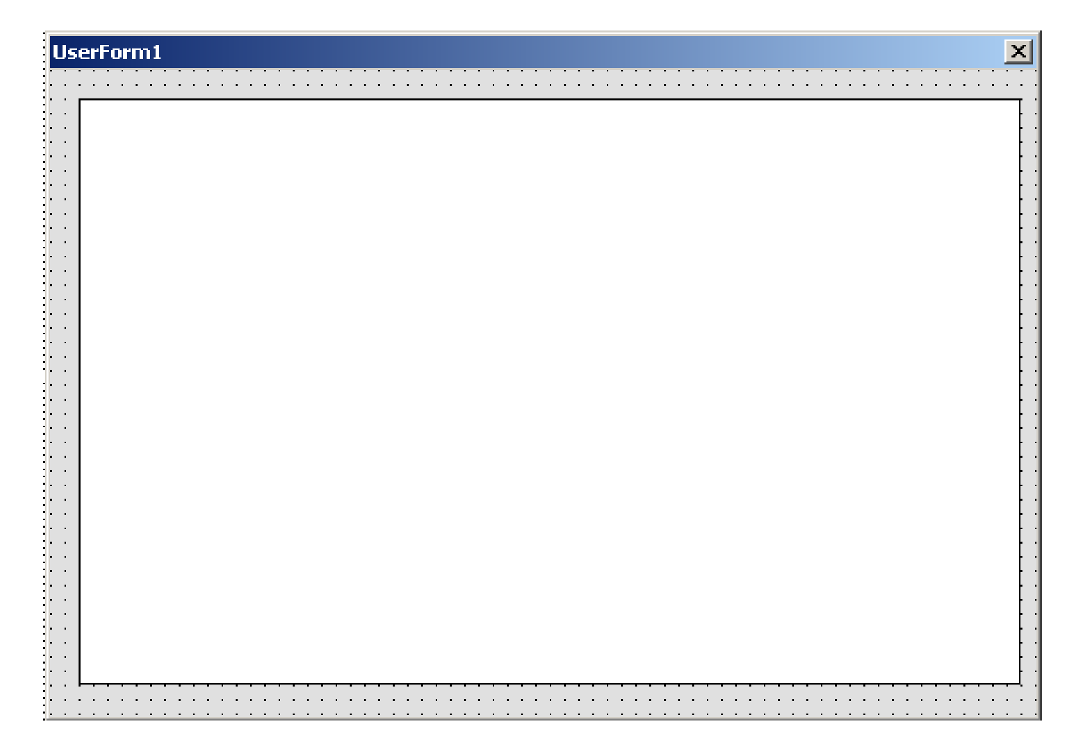
Microsoft Office Chart

qui ajoutera le bouton suivant dans la *Toolbox*:

Pour l'exemple, nous allons utiliser le tableau suivant:

	A	B	C	D	E	F	G	H
1		LUNDI	MARDI	MERCREDI	JEUDI	VENDREDI	SAMEDI	DIMANCHE
2	24.10.2005	67	77	87	49	33	74	45
3	25.10.2005	88	39	59	77	39	34	58
4	26.10.2005	49	28	38	46	58	55	30
5	27.10.2005	95	79	39	67	83	28	23
6	28.10.2005	28	55	95	76	33	44	40

Ensuite, nous ajoutons un contrôle *ChartSpace1* à un *Userform*:



et dans l'initialisation du formulaire, nous ajoutons le code suivant:

```
Private Sub UserForm_Initialize()

    Dim C
    Dim Cht As OWC.WCChart
    Set C = ChartSpace1.Constants
    Set Cht = ChartSpace1.Charts.Add
    Dim i, x, j As Integer
    Dim Tableau(30), Plage(30)

    'supression des series existantes dans le ChartSpace au besoin
    For i = Cht.SeriesCollection.Count To 1 Step -1
        Cht.SeriesCollection.Delete i - 1
    Next i

    'on compte le nombre de colonne (futures abscisses) de la table
    'on commence à B1 car A1 est vide
    For i = 1 To Range("B1").End(xlToRight).Column
        Tableau(i) = Cells(1, i)
    Next i

    'type de graphique:Barres
    Cht.Type = C.chChartTypeColumnClustered

    'On itère sur le nombre de séries (lignes du tableau)
    For j = 0 To Range("A2").End(xlDown).Row
        'pour chaque linge on ajoute une série
        Cht.SeriesCollection.Add

        'on remplit un tableau avec les données (ordonnées de la série en
cours
        For i = 1 To Range("B1").End(xlToRight).Column
            Plage(i) = Cells(j + 2, 1 + i)
        Next i
        With Cht
            .SetData C.chDimCategories, C.chDataLiteral, Tableau
```

```
            .SeriesCollection(x).SetData C.chDimValues, C.chDataLiteral,
Plage
            .SeriesCollection(x).Interior.Color = 50000 * (j + 1)
        End With
        x = x + 1
        'on vide le tableau pour la prochaine série
        Erase Plage
    Next j

End Sub
```

15.4.21 Contrôle Media Player

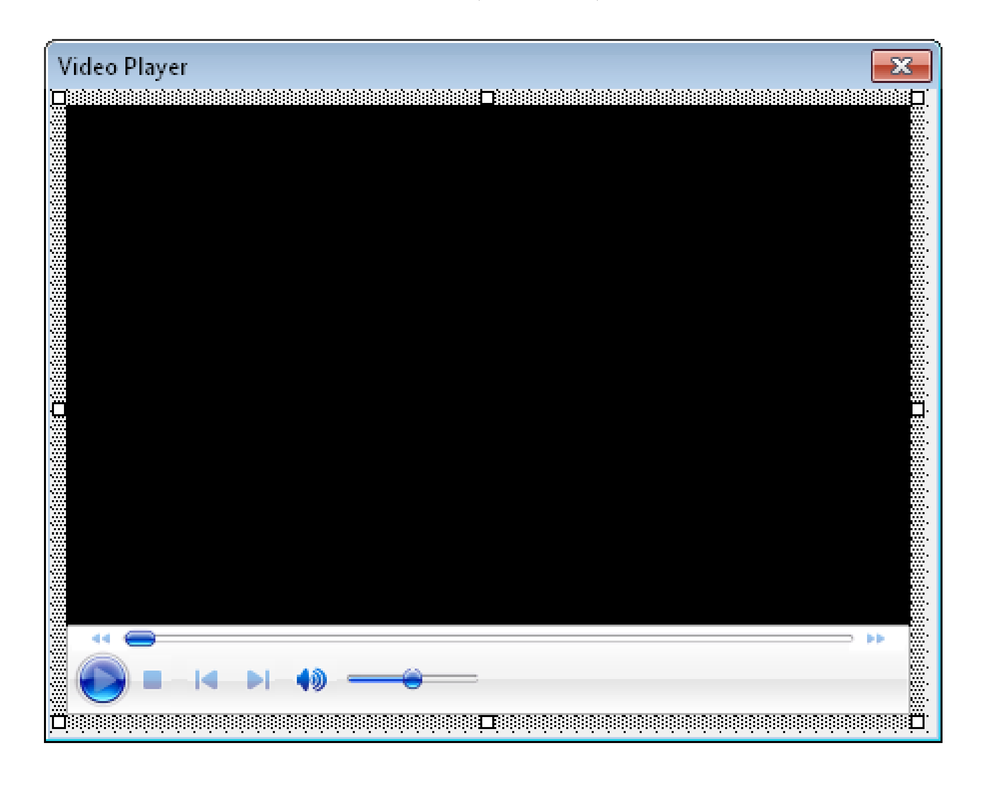
Nous allons ici nous intéresser à l'insertion d'un graphique Excel simple dans un *Userform* ce qui peut être très utils dans des logiciels comme Excel, Access ou particulièrement Project.

Pour voir un exemple il faut d'abord ajouter le contrôle:

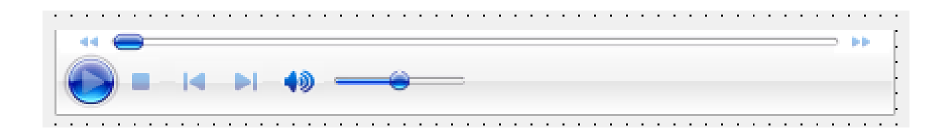
Windows Media Player

qui ajoutera le bouton suivant dans la *Toolbox*: 

Et en ajoutant le contrôle dans un formulaire (userform) de votre choix, vous aurez:



En jouant avec la hauteur du contrôle on peut le réduire à:



Afin de n'avoir qu'un lecteur de musique (mp3) typiquement.

Ensuite, il suffit au chargement du formulaire, ou sur clic d'un bouton, ou sur une listbox ou sur tout autre événement de votre choix, de mettre le code suivant:

```
WindowsMediaPlayer1.settings.AutoStart = False
WindowsMediaPlayer1.URL = "C:\Users\Isoz Vincent\Downloads\HubbleWMV.wmv"
'Pour spécifier le début de la séquence
WindowsMediaPlayer1.Controls.CurrentPosition = 5 'secondes
'Pour couper le son
WindowsMediaPlayer1.settings.Mute = True
'Régler le niveau du son ....Settings.Volume=50
'Pour jouer en boucle
WindowsMediaPlayer1.settings.setMode "loop", True
```

et évidemment... vous trouverez beaucoup plus d'information sur Google!



Formulaires (exercices)

```
'*******************************
'Créateur: Vincent ISOZ
'Dernière modification: 28.11.2003
'Nom procédure: -
'Commentaires: Utiliser l'ActiveX "Control Calendar 9.0" pour une saisie ergonomique des dates dans un userform
'*******************************
```

Pour accéder au *Control Calendar* faites un clic droit sur la palette d'outils (Control Toolbox) et choisissez *Contrôles supplémentaires*. Ensuite, cochez la case *Contrôle Calendar…* et ajoutez ce contrôle et d'autres à un nouveau Userform tel que:

Ensuite, double cliquez sur le calendrier et saisissez-y le code suivant:

```
Private Sub Calendar1_Click()
    TextBox1.Value = Calendar1.Value
End Sub
```

Et voilà !

En tant qu'exercice, créez un bouton sur Userform 1 qui ouvre un nouvel Userform (Userform2) avec le calendrier. Lorsque l'utilisateur clique sur une date du calendrier se trouvant sur le Userform2, la date en question doit être retournée au TextBox1 de Userform1 (le code est très semblable).

```
'*******************************
'Créateur: Vincent ISOZ
'Dernière modification: 29.10.2003
'Nom procédure: copieword()
'Commentaires: Cettre procédure fonctionne avec le userform "listefeuilles"
'permettant une interaction dynamique entre la classeur et l'utilisateur
'et utilisant une liste à choix multiple
'*******************************

Sub listeFeuilles()

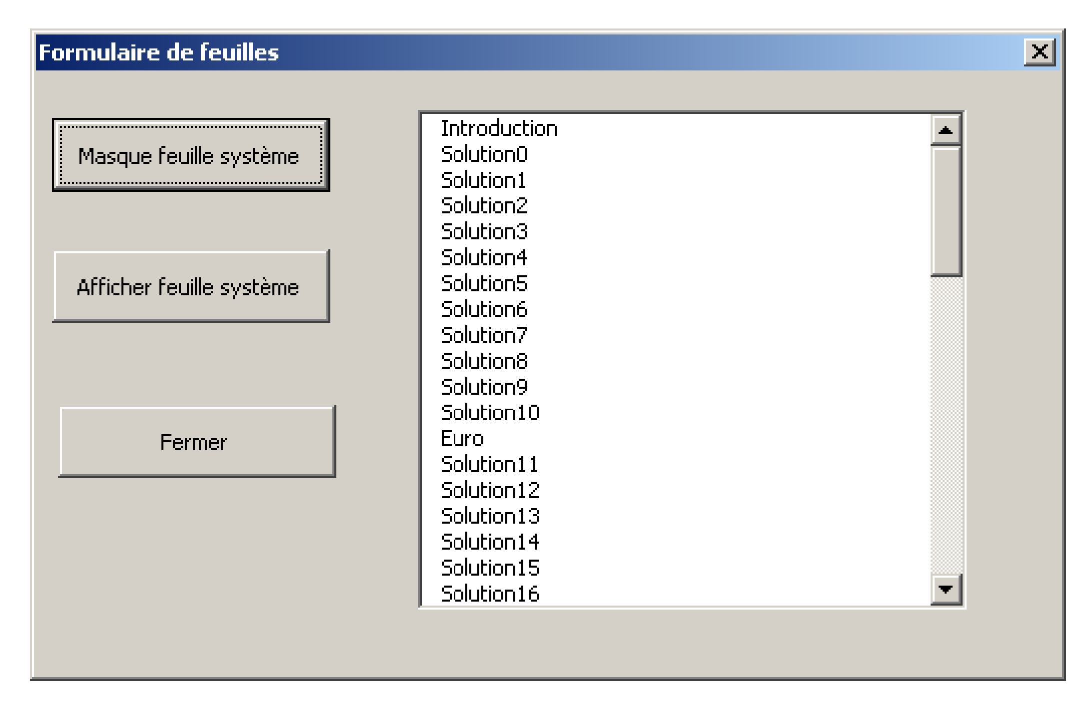
    Dim chemin, dept As String
```

```
    Dim i As Long
    chemin = ActiveWorkbook.Path
    'Nous créeons une variable ainsi, au besoin nous pouvons demander
    'dans une inputbox le chemin à l'utilisateur
    chemin = chemin & "\dossierexcel"
    'Il faut choisir une feuille du classeur dans laquelle mettre
    'la liste des fichiers trouvés
    Sheets("liste_fichiers").Select

    'File search ne fonctionne plus depuis Excel 2007
    With Application.FileSearch
        'La ligne ci-dessous va permette (on ne sait jamais)
        'de réinitialiser tous les paramètres de recherche
        'd'une éventuelle recherche antérieure
        .NewSearch
        'l'endroit où doit cherche Excel (ou word)
        .LookIn = chemin
        'est-ce qu'il doit aller regarder aussi dans les sous-dossiers
        .SearchSubFolders = False
        'types de fichiers (même principe que l'explorateur)
        .Filename = "*.*"
        'la méthode execute va permette de controler s'il y au moins
        'un fichier de trouvé
        If .Execute() > 0 Then
            MsgBox "Il y a " & .FoundFiles.Count & " fichier(s) trouvé(s)."
            For i = 1 To .FoundFiles.Count
                'nous affichons les extensions de fichiers
                dept = Right(.FoundFiles(i), 3)
                Cells(i, "A") = .FoundFiles(i)
                Cells(i, "B") = dept
            Next i
        Else
            MsgBox "Pas de fichiers"
        End If
    End With

End Sub

'*******************************
'Le userform utilisé précédemment (évidemment la ListBox peut être
différente):
```




```
'et le code qui se trouve derrière

'on remplit la listbox qui se nomme "fc_listfeuil"
'nous aurions très bien pu faire cela sur l'événement Userform_intialize()
Private Sub UserForm_Activate()
    Dim z_cpt As Variant
    fc_listfeuil.Clear

    fc_listfeuil.MultiSelect = fmMultiSelectMulti
    For z_cpt = 1 To Sheets.Count
        'Pour ajouter un Item manuellement taper:
nomListebox.Additem("element")
        fc_listfeuil.AddItem Sheets(z_cpt).Name
    Next z_cpt

End Sub

'le bouton Afficher Feuille se nomme "affeuil"
Private Sub affeuil_Click()
    Dim z_cpt As Byte
    Dim z_result As String

    For z_cpt = 0 To fc_listfeuil.ListCount - 1
        If fc_listfeuil.Selected(z_cpt) = True Then
            z_message = "masquer " & fc_listfeuil.List(z_cpt)
            z_result = MsgBox(z_message, vbOKCancel, "Affichage des
feuilles")
            If z_result = vbOK Then
                Sheets(fc_listfeuil.List(z_cpt)).Visible = True
            End If
        End If
    Next z_cpt
End Sub
```

```
'le bouton Masquer Feuille se nomme "maskfeuil"

Private Sub fb_maskfeuil_Click()
    Dim z_cpt As Byte
    Dim z_result As String

    For z_cpt = 0 To fc_listfeuil.ListCount - 1
        If fc_listfeuil.Selected(z_cpt) = True Then
            z_message = "masquer " & fc_listfeuil.List(z_cpt)
            z_result = MsgBox(z_message, vbOKCancel, "Masquage des
feuilles")
            If z_result = vbOK Then
                Sheets(fc_listfeuil.List(z_cpt)).Visible = False
            End If
        End If
    Next z_cpt
End Sub

'si l'utilisateur souhaite masquer le userform
Private Sub listesfeuilles_Click()
    listeFeuilles.Hide
    Unload listeFeuilles
End Sub


'*******************************
'Créateur: Vincent ISOZ
'Dernière modification: 29.10.2003
'Nom procédure: prc_class(), montre_user()
'Commentaires: les deux procédures suivantes sont nécessaires pour
l'exemple du 'module de classe " ClsZoneLabel" qui utilise aussi le
Userform "Class"
'Ce programme ne fait que de montrer comment une classe agit tel quel
(comme une classe 'quoi…) sur un groupe de label figurant sur un formulaire
'*******************************

'z_label est un tableau déclarant une nouvelle instance de la classe
ClsZoneLabel
'il est obligatoire de définir cette variable en tant que variable globale

'ClsZoneLabel est le nom du module de Classe utilisé dans cet exemple
Dim z_label() As New ClsZoneLabel

'fait référence à la classe du même nom

Sub prc_class()

'il faut balayer l'ensemble des contrôles du formulaire, et en fonction
'de leur type, les affecter à la classe
Dim z_ctl As Control
'permet de compter les champs de type label
Dim z_labelcount As Integer

z_labelcount = 0

For Each z_ctl In Class.Controls
    If TypeName(z_ctl) = "Label" Then
        z_labelcount = z_labelcount + 1
        'on redimensionne la classe qui se comporte donc comme un tableau
        ReDim Preserve z_label(1 To z_labelcount)
        'on initiale le textbox en cours relativement à la classe
```

```
        Set z_label(z_labelcount).LabelGroup = z_ctl
    End If
Next z_ctl
End Sub

'La procédure a exécuter!
Sub montreuser()
    prc_class
    'on ouvre le userform Class utilisant le module de classe
    Class.Show
End Sub

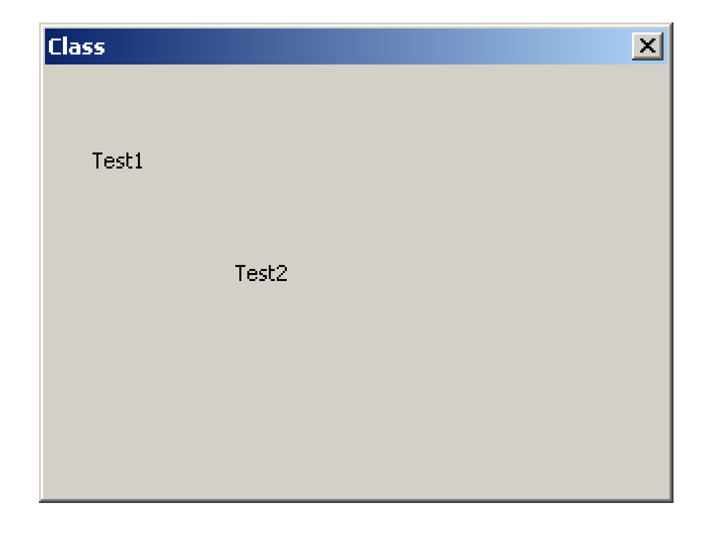
'******************************
'Voici le contenu du module de classe utilisé ci-dessus
'Le module se nomme ClsZoneLabel
'******************************

Option Explicit

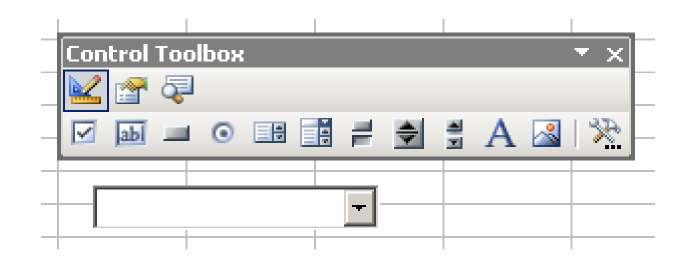
'Nous déclarons de manière Public les événements sur les objets de type
Label
'que nous groupons sous le nom LabelGroup
Public WithEvents LabelGroup As MSForms.Label

'Faire en sorte que n'importe quel type de label se déplace
Private Sub LabelGroup_MouseDown(ByVal Button As Integer, ByVal Shift As
Integer, ByVal X As Single, ByVal Y As Single)
'si on clique avec le bouton droit de la souris sur un label
If Button = 1 Then
    LabelGroup.Left = LabelGroup.Left + 20
    LabelGroup.Top = LabelGroup.Top + 20
End If
End Sub

'***********************
'Le formulaire nommé "Class" en question qui ne contient aucun code mis à
part quelques labels créés au préalable pour l'exemple.
```

Autre chose, lorsque vous créez une listbox de type **Control** (Toolbox):



sur une feuille MS Excel directement, voici le code événementiel nécessaire pour la remplir:

```
Private Sub Combox1_GotFocus()
     'Attention à faire un clear avant de charger la liste
     ComboBox1.AddItem "V.B.A."
     ComboxBox1.AddItem "VB.Net"
     ComboxBox1.AddItem "ASP.Net"
     ComboxBox1.AddItem "C#"
End Sub
```



16. ÉVÉNEMENTIEL

```
'********************************
'Créateur: Vincent ISOZ
'Dernière modification: 29.10.2003
'Commentaire: Cette procédure événementielle permet d'ajouter
automatiquement des 'boutons, menus et barres d'outils dans Excel 'à
l'installation d'un add-in (macro 'complémentaire)
'Evidemment, on va lier une routine à ces différents éléments afin qu'ils
aient une utilité
'Pour plus d'infos sur les icônes, voir la fin du présent document
'********************************

Sub Workbook_AddinInstall()

    Application.CommandBars.Add "Sciences.ch", msoBarTop
    Application.CommandBars("Sciences.ch").Visible = True
    Application.CommandBars("Sciences.ch").Protection = msoBarNoMove

    Set menu1 = Application.CommandBars("Worksheet Menu
Bar").Controls.Add(msoControlPopup)
    Set menu2 =
Application.CommandBars("Sciences.ch").Controls.Add(msoControlPopup)

    Set Opt11 = menu1.Controls.Add(msoControlButton, 47)
    Set Opt12 = menu1.Controls.Add(msoControlButton, 1950)
    Set Opt13 = menu1.Controls.Add(msoControlButton, 542)
    Set Opt21 = menu2.Controls.Add(msoControlButton, 4)
    Set Opt22 = menu2.Controls.Add(msoControlButton, 3)
    Set Opt23 = menu2.Controls.Add(msoControlButton)
    Set Opt31 =
Application.CommandBars("Data").Controls.Add(msoControlButton, 2950,
Before:=2)

    Set bouton1 =
Application.CommandBars("Sciences.ch").Controls.Add(msoControlButton, 47)
    Set bouton2 =
Application.CommandBars("Sciences.ch").Controls.Add(msoControlButton, 1950)

    menu1.Caption = "Mon Entreprise SA"
    menu2.Caption = "Sciences.ch"

    bouton1.Caption = "Effacer les données"
    bouton2.Caption = "Saisir les données"
    bouton1.OnAction = "subhello"
    bouton2.OnAction = "subhello"

    Opt11.OnAction = "subhello"
    Opt12.OnAction = "subhello"
    Opt13.OnAction = "subhello"
    Opt13.ShortcutText = "Ctrl+L"
    Opt21.OnAction = "subhello"
    Opt22.Caption = "Never save it again"
    Opt22.OnAction = "subhello"
    Opt22.ShortcutText = "Ctrl+S"
    Opt23.Caption = "Hi nice to meet you"
    Opt23.OnAction = "subhello"
    Opt31.Caption = "Hello it works"
```

```
    Opt31.OnAction = "subhello"

End Sub

'*******************************
'Créateur: Vincent ISOZ
'Dernière modification: 29.10.2003
'Commentaire: lorsque l'utilisateur désinstalle la macro complémentaire, nous effaçons tous 'les menus et boutons
'*******************************

Private Sub Workbook_AddinUninstall()
    Application.CommandBars("Worksheet Menu Bar").Controls(11).Delete
    Application.CommandBars("Sciences.ch").Delete
    Application.CommandBars("Data").Controls(2).Delete
End Sub

'*******************************
'Créateur: Vincent ISOZ
'Dernière modification: 29.10.2003
'Commentaire: L'activation. Cela vous dit quelque chose…
'*******************************

Private Sub Workbook_Open()
    MsgBox "Vous avez jusqu'à au 25 février 2002 pour commander une nouvelle licence"
    reste = 25 - Day(Date)
    MsgBox "Il vous reste: " & reste & "jour(s)"
    If Day(Date) = 25 Then
        Workbook.Close
    End If
End Sub

'*******************************
'Créateur: Vincent ISOZ
'Dernière modification: 29.10.2003
'Commentaire: Très pratique pour ranger les fonctions que l'on a créées…
'*******************************

Private Sub Workbook_Open()
    Welcome.Show
    'Welcome.Show vbModeless pour pouvoir écrire derrière la boîte de dialogue
    'On range la fonction "utilisateur" dans la catégorie 9 des fonctions
    Application.MacroOptions Macro:="utilisateur", Category:=9
    0 Toutes
    1 Finances
    2 Date & Heure
    3 Math & Trigo
    4 Statistiques
    5 Recherche et Matrices
    6 Base de données
    7 Texte
    8 Logique
    9 Information
    10 Commandes
    11 Personnalisées
    12 Contrôle de Macros (optionnel)
    13 DDE/Externes (optionnel)
    14 Définies par l'utilisateur
    15 Scientifiques
```

```
End Sub

'******************************
'Créateur:?
'Dernière modification: 12.09.2010
'Commentaire: Permet de protéger une feuille tout en laissant les macros
faire leur job et en même temps permet d'utiliser le mode plan qui par
défaut est bloqué normalement avec la protection
'******************************

Private Sub Workbook_Open()

    'doit obligatoirement s'executer à chaque ouverture du classeur
    'sinon excel ne se souviendra pas que userinterface=True
    With Worksheets("Mode Plan")
        'userinterface à true permet d'exécuter n'importe quelle macro
sinon quoi elles n'arriveront pas à faire ce qu'elles doivent sur la
feuille
        .Protect Password:="toto", userinterfaceonly:=True
        .EnableOutlining = True
    End With
End Sub

'******************************
'Créateur: Vincent ISOZ
'Dernière modification: 04.08.2004
'Commentaire: On crée un nouveau bouton dans le menu contextuel de la
souris
' A vous de faire que cela ce fasse à l'ouverture d'un classeur spécifique
ou d'Excel en général
'******************************

Sub AddMenuContextModeDesign()
'Cette procedure va ajouter le bouton dans le menu contextuel
With Application.CommandBars("Cell").Controls.Add(msoControlButton)
   .Caption = "Affichage Plein Ecran"
   .BeginGroup = False
   .FaceId = 178
   'appelle la procedure à effectuer lors du clic sur le bouton
   .OnAction = "PleinEcran"
End With
'Pour supprimer le bouton au besoin:
'Application.CommandBars("Cell").Controls("Affichage Plein Ecran").Delete
Range("F29").Select 'ceci juste pour les besoins de la démo...
End Sub

Sub PleinEcran()
     'Procédure appelée lors du "OnAction" précédent
     Application.CommandBars.FindControl(ID:=178).Execute
End Sub
```

17. OBJECT LINKED AND EMBEDDED (OLE)

Dans ce chapitre nous allons rapidement présenter l'architecture logicielle d'OLE.

17.1 Architecture

Sous l'appellation OLE, on désigne en fait toute l'architecture logicielle de communication inter-applications de Microsoft, allant de la spécification COM aux contrôles ActiveX. OLE c'est en fait un certain nombre de services que le programmeur peut exploiter dans ses applications Windows courramment abrégée ADO (*ActiveX Data Objects*).

Remarque: ADO remplace DAO (Data Access Objects) qui existait dans Excel 97. Ainsi, vous trouvez DAO et ADO dans MS Excel 2000 et supérieures mais vous ne trouvez pas ADO dans MS Excel 97.

La figure ci-dessous précise les différents services disponibles dans l'environnement OLE.

ActiveX / DCOM
OLE Automation
Liaison et encapsulation
Documents composés
Stockage d'objets composés
COM

Cette architecture est basée sur le modèle COM (*Component Object Model*). Nous trouvons ensuite un ensemble de service concernant les documents composés (*Compound Objects*) permettant l'échange de données entre documents. Ces services sont représentés par trois couches dans la figure précédente. Au dessus, on distingue le service *OLE Automation* qui permet à un programme d'exécuter des méthodes d'un objet serveur. Enfin, la dernière brique de cet édifice est ActiveX qui correspond en fait à des contrôles OLE renommés début 96 par Microsoft. On distingue également à ce niveau le modèle *DCOM*, version distribuée du modèle COM.

Nous n'allons détailler que les principaux services cités ci-dessus.

17.2 COM (Component Object Model)

Le *Component Object Model* est, comme nous l'avons dit, le format des applications OLE, leur permettant ainsi de pouvoir communiquer entre elles. Ce format est indépendant du langage de programmation choisi, l'essentiel est que l'exécutable obtenu respecte le modèle.

Notons que nous parlons d'exécutable mais en réalité il s'agit plutôt de bibliothèques de fonctions exécutables par plusieurs applications OLE.

Il est important de remarquer qu'OLE est un environnement basé sur des objets client/serveur. Un objet, hébergé par un serveur (bibliothèque *DLL*, application etc.), peut être utilisé par plusieurs programmes clients simultanément.

Du point de vue du programmeur, l'élément de base en programmation OLE est l'*interface*, *classe abstraite* ou *classe virtuelle*. Il s'agit exactement de ce que les programmeurs POO utilisent: une interface est une classe dont toutes les méthodes sont abstraites. Le programmeur va donc devoir implémenter chacune de ces méthodes pour une interface donnée. L'accès à un objet se fait via des pointeurs sur ses interfaces.

L'interface de base, qu'il est obligatoire d'implémenter pour un objet, est l'interface IUnknown. Cette interface est disponible pour tous les objets OLE/COM.

L'interface IUnknown possède, entre autres, une méthode appelée QueryInterface() qui permet à un programme utilisant l'objet considéré de prendre connaissance de l'existence d'une interface implémentée dans l'objet. Si l'interface demandée existe, la méthode QueryInterface() renvoie un pointeur sur celle-ci afin que l'objet demandeur puisse l'utiliser. Chaque interface d'un objet est identifiée de façon unique par un *GUID* (*Globally Unique IDentifier*) attribué par Microsoft pour éviter tout conflit d'identifiant. Ce dernier, codé sur 128 bits, est stocké dans la *base des registres* de Windows. Dans cette base, pour chaque GUID, on trouve le nom et la localisation du serveur (DLL par exemple) fournissant l'objet dont on veut accéder à l'une de ses interfaces. On trouve également pour chaque interface un certain nombre d'informations telles que sa version, les types de données reconnus, etc.
A chaque fois que l'interface d'un objet est utilisée par un client, un compteur de référence est incrémenté par ce client, via l'appel d'une fonction AddRef(). Lorsque le client n'utilise plus cette interface, il appelle la méthode Release() qui décrémente le compteur. Dès que la valeur de ce compteur est nulle, la mémoire utilisée par l'objet considéré est libérée automatiquement.

Une autre interface de base est IClassFactory. Cette interface permet de créer un objet, via une méthode appelée CreateInstance() à laquelle on indique en argument le GUID de l'objet à instancier. Cette méthode renvoie un pointeur vers l'interface IUnknown que nous venons de présenter, permettant ainsi d'utiliser l'objet considéré. Précisons par ailleurs que le mécanisme que nous venons d'évoquer est automatisé par l'appel de la fonction COM CoGetClassObject() à laquelle on fournit un GUID qui renvoie donc le pointeur vers l'interface IUnknown correspondante.

Notons enfin qu'il n'existe pas de mécanisme d'héritage pour les interfaces du modèle COM. On utilise à la place soit des *agrégations* (désignation d'un agrégat de plusieurs objets désigné par une seule et même interface IUnknown) soit des *délégations* (où une interface en utilise une autre). Nous n'en dirons pas plus sur ces mécanismes.

17.3 OLE Automation

OLE Automation est un service très puissant qui permet de manipuler une application en appelant dynamiquement les méthodes des objets, respectant le format COM, qu'elle contient.

Cette manipulation est typiquement faite via un langage de scripts tel que *VBA*, mais ce n'est pas une obligation.

En fait, OLE Automation permet à une application de découvrir dynamiquement les objets et méthodes disponibles dans une autre application afin de pouvoir les utiliser. OLE Automation permet également de construire des serveurs d'objets qui sont capables de savoir quelle méthode de quel objet appeler suite à une requête d'une application cliente.

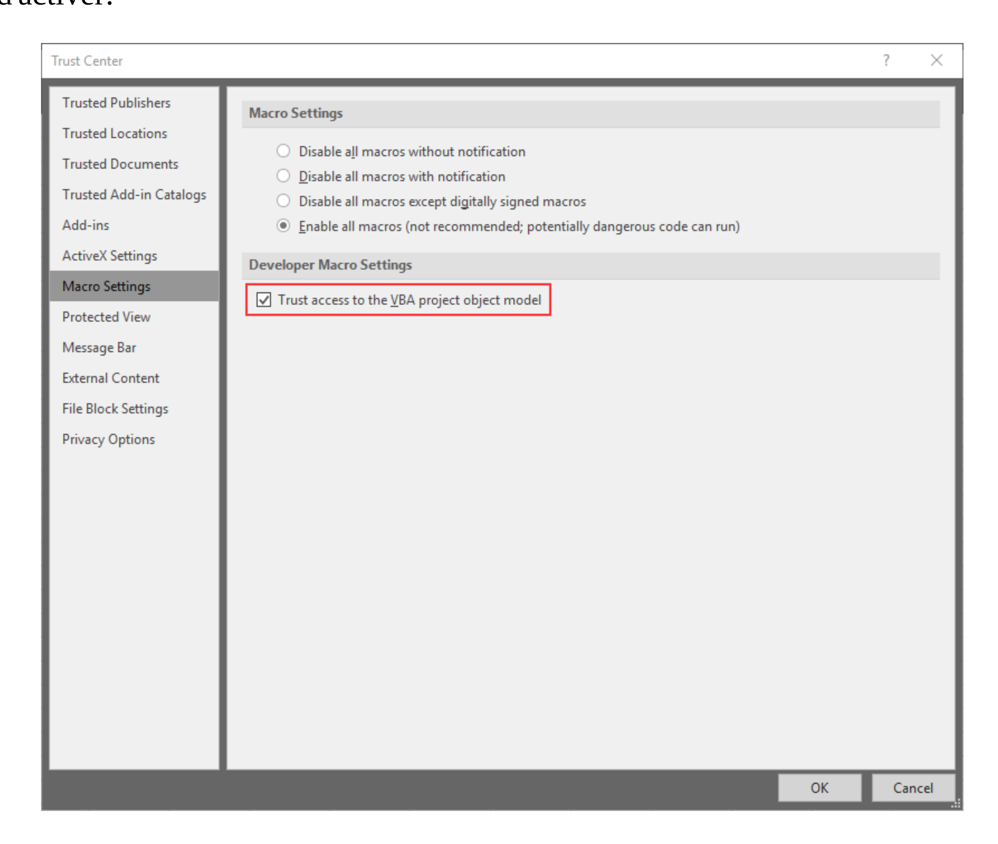
On définit dans OLE Automation des objets automates (*Automation Objects*) qui pourront être manipulés à l'aide de langages de scripts tels que V.B.A. ou être invoqués dynamiquement. Ces applications clientes sont appelées des contrôleurs d'automates.

Nous avons vu plus haut que pour tout objet, on devait obligatoirement implémenter l'interface IUnknown. Dans le cas d'un objet automate, on doit également implémenter de façon obligatoire une autre interface, appelée IDispatch et qui permet les invocations dynamiques dont nous venons de parler. Cette interface est utilisée afin d'obtenir le nom de la fonction à appeler tandis que la signature de celle-ci est extraite d'une *librairie de types* des fonctions des objets automates. Cette librairie est généralement stockée dans un DLL ou un fichier annexe.

Notons enfin que la signature de ces fonctions est décrite grâce à un langage appelé *ODL* pour *Object Description Language*.

17.3.1 Gérer les références externes

Avant tout chose, il est évident qu'on peut ajouter les références externes à main. Mais lorsqu'on vent des solutions à des clients, ce type d'approche n'est pas acceptable. Une méthode consiste alors à attaquer ces dernières grâche leur GUID. Mais d'abord il ne faut pas oublier d'activer:



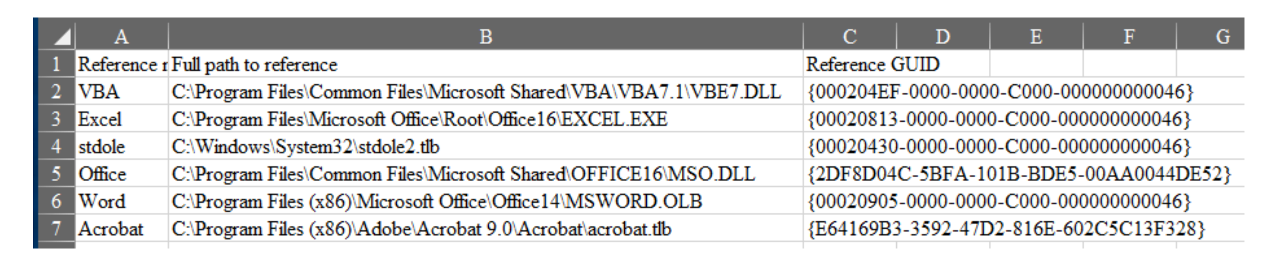
Une fois ceci activé, voyons déjà le code permettant d'afficher la liste des GUID rapidement (source: http://www.vbaexpress.com/kb/getarticle.php?kb_id=278):

```
Option Explicit

Sub ListReferencePaths()
     'Macro purpose:  To determine full path and Globally Unique Identifier
(GUID)
     'to each referenced library.  Select the reference in the
Tools\References
     'window, then run this code to get the information on the reference's
library

    On Error Resume Next
    Dim i As Long
    With ThisWorkbook.Sheets(1)
        .Cells.Clear
        .Range("A1") = "Reference name"
        .Range("B1") = "Full path to reference"
        .Range("C1") = "Reference GUID"
    End With
    For i = 1 To ThisWorkbook.VBProject.References.Count
        With ThisWorkbook.VBProject.References(i)
            ThisWorkbook.Sheets(1).Range("A65536").End(xlUp).Offset(1, 0) =
.Name
            ThisWorkbook.Sheets(1).Range("A65536").End(xlUp).Offset(0, 1) =
.FullPath
            ThisWorkbook.Sheets(1).Range("A65536").End(xlUp).Offset(0, 2) =
.GUID
        End With
    Next i
    On Error GoTo 0
End Sub
```


Ce qui donnera typiquement:



	A	B	C	D	E	F	G
1	Reference r	Full path to reference	Reference GUID				
2	VBA	C:\Program Files\Common Files\Microsoft Shared\VBA\VBA7.1\VBE7.DLL	{000204EF-0000-0000-C000-000000000046}				
3	Excel	C:\Program Files\Microsoft Office\Root\Office16\EXCEL.EXE	{00020813-0000-0000-C000-000000000046}				
4	stdole	C:\Windows\System32\stdole2.tlb	{00020430-0000-0000-C000-000000000046}				
5	Office	C:\Program Files\Common Files\Microsoft Shared\OFFICE16\MSO.DLL	{2DF8D04C-5BFA-101B-BDE5-00AA0044DE52}				
6	Word	C:\Program Files (x86)\Microsoft Office\Office14\MSWORD.OLB	{00020905-0000-0000-C000-000000000046}				
7	Acrobat	C:\Program Files (x86)\Adobe\Acrobat 9.0\Acrobat\acrobat.tlb	{E64169B3-3592-47D2-816E-602C5C13F328}				

Voyons maintenant comment charger par exemple la référence *Acrobat* par rapport à son GUID:

```
Sub AddReference()
     'Macro purpose:  To add a reference to the project using the GUID for
the
     'reference library

    Dim strGUID As String, theRef As Variant, i As Long

     'Update the GUID you need below.
    strGUID = "{E64169B3-3592-47D2-816E-602C5C13F328}"

     'Set to continue in case of error
    On Error Resume Next
```

```
     'Remove any missing references
    For i = ThisWorkbook.VBProject.References.Count To 1 Step -1
        Set theRef = ThisWorkbook.VBProject.References.Item(i)
        If theRef.isbroken = True Then
            ThisWorkbook.VBProject.References.Remove theRef
        End If
    Next i

     'Clear any errors so that error trapping for GUID additions can be
evaluated
    Err.Clear

     'Add the reference
    ThisWorkbook.VBProject.References.AddFromGuid _
    GUID:=strGUID, Major:=1, Minor:=0

     'If an error was encountered, inform the user
    Select Case Err.Number
    Case Is = 32813
         'Reference already in use.  No action necessary
    Case Is = vbNullString
         'Reference added without issue
    Case Else
         'An unknown error was encountered, so alert the user
        MsgBox "A problem was encountered trying to" & vbNewLine _
        & "add or remove a reference in this file" & vbNewLine & "Please
check the " _
        & "references in your VBA project!", vbCritical + vbOKOnly,
"Error!"
    End Select
    On Error GoTo 0
End Sub
```


17.3.2 Automation MS Word

Voici plusieurs exemples d'utilisation d'OLE avec MS Word:

17.3.2.1 Copie d'un tableau MS Excel dans MS Word avec liaison

Il s'agit d'un code qui copie et colle (entre autres) un tableau Excel avec liaison dans un document Word (de nombreux autres exemples OLE Automation se trouvent déjà dans les exercices sur les procédures):

```
Sub AutomateWord()
'Ne pas oublier d'ajouter la référence Word au projet!

    Dim appWord As New Word.Application
    Dim docWord As New Word.Document
    Dim rng As Range

    'Ajoute un nouveau document
    With appWord
        .Visible = True
        Set docWord = .Documents.Add
        .Activate
    End With
```

```
    'ou pour agir un document sans l'ouvrir on utilise la fonction
GetObject
    'Set docWord= GetObject("c:\perso\temp\test.doc")

    With appWord.Selection
        'ajoute une ligne de titre et la met en forme
        .TypeText Text:="Résultat des ventes de 2003"
        .HomeKey Unit:=wdLine
        .EndKey Unit:=wdLine
        .ParagraphFormat.Alignment = wdAlignParagraphCenter
        .Font.Size = 18
        With .Font
             .Name = "Verdana"
             .Bold = True
             .Italic = False
             .SmallCaps = True
        End With
        'Copie le tableau Excel dans le presse-papiers
        Range("A1:D10").Copy
        'Colle le tableau dans word avec liaison
        .EndKey Unit:=wdLine
        .TypeParagraph
        .TypeParagraph
        .PasteSpecial link:=True, DataType:=wdPasteOLEObject,
Placement:=wdInLine, DisplayAsIcon:=False
        'Copie le graphique dans Word avec liaison
        ActiveSheet.ChartObjects(1).Activate
        ActiveChart.ChartArea.Select
        ActiveChart.ChartArea.Copy
        'Colle le graphique dans Word avec liaison
        .TypeParagraph
        .TypeParagraph
        .PasteAndFormat (wdChartLinked)
    End With

    With docWord
        'Enregistre le document word dans le même dossier que le classeur
Excel
        .SaveAs ThisWorkbook.Path & "\Resultat_2003.doc",
Allowsubstitutions:=True
        'Aperçu du résultat dans Word
        .PrintPreview
    End With
    'Réinitialise l'objet
     appWord.Quit
     Set appWord = Nothing

End Sub
```


17.3.2.2 Copies de tableaux MS Word dans MS Excel

Le code ci-dessous parcoure tous les fichier dans un dossier donné et rappatrie le contenu du *x-ème* tableau du document MS Word dans la feuille MS Excel en cours:

```
Sub ReadExternalWordTables()
    Set WordApp = CreateObject("Word.Application")
    'WordApp.Visible = True

   'Here put your path where you have your documents to read:
```

```
    myPath = "C:\MonDossier\"
    myFile = Dir(myPath & "*.docx")

    xlRow = 1
    Do While myFile <> ""
       Set WordDoc = WordApp.Documents.Open(myPath & myFile)
       xlCol = 0
       i = 1
       For Each t In WordApp.activedocument.tables
         'to retrieve only the second table
         If i = 2 Then
              For Each r In t.Rows
                 For Each c In r.Range.Cells
                    myText = c
                    myText = Replace(myText, Chr(13), "")
                    myText = Replace(myText, Chr(7), "")
                    xlCol = xlCol + 1
                   ActiveSheet.Cells(xlRow, xlCol) = myText
                 Next c
                 xlRow = xlRow + 1
                 xlCol = 0
              Next r
          End If
          i = i + 1
       Next t
       WordDoc.Close
       Set WordDoc = Nothing
       myFile = Dir
    Loop
    Set WordApp = Nothing
End Sub
```


17.3.2.3 Récupération de la valeur d'un signet dans MS Word

```
Sub RecupSignet()
     Dim Wd As Word.Application
     Set Wd = New Word.Application
     With Wd
          .Visible = False
          .documents.Open ("c:\Mes documents\Test\test.doc")
          .Selection.GoTo What:=wdGoToBookmark, Name:="SIGNET1"
          MsgBox .Selection.Text
          .Quit False
     End With
End Sub
```

17.3.2.4 Import de tous les textes d'un document MS Word

Très utile pour ceux qui font du Text Mining: l'import de tous les paragraphes d'un document Microsoft Word:

```
Sub GetTextFromWord()

    file = "C:\tmp\Test.docx"
    Set WordApp = CreateObject("Word.Application")
    WordApp.Visible = True
    Set WordDoc = WordApp.Documents.Open(file)

    Set myRange = WordDoc.Range
```

```
    documentText = myRange.Text

    i=1
    For Each para In WordDoc.Paragraphs
            title = para.Range.Text
            Application.ActiveWorkbook.Worksheets("Sheet1").Cells(i,
2).Value = title
            i = i + 1
    Next para

    WordDoc.Close
End Sub
```

17.3.3 Automation MS PowerPoint

Voici plusieurs exemples d'utilisation d'OLE avec MS PowerPoint:

17.3.3.1 Copie d'un graphique MS Excel dans un nouveau diaporama MS PowerPoint

Un autre exemple d'automation dans lequel nous copions un graphique dans une présentation powerpoint créée à la volée:

```
Sub AutomatePWP()
'Ne pas oublier d'ajouter la référence PowerPoint au projet!

    Dim appPW As New PowerPoint.Application
    Dim nwSlide As PowerPoint.Slide
    appPW.Presentations.Add
    appPW.Visible = True
    'on ajoute une diapo dans le diaporama précédemment créé
    Set nwSlide = appPW.ActivePresentation.Slides.Add(1, ppLayoutChart)
    'on définit la feuille sur laquelle se trouve le graphique à coller
dans PowerPoint
    Set shtExcel = ActiveWorkbook.Sheets(1)

    With nwSlide
        .Shapes(1).TextFrame.TextRange.text = "V.B.A. c'est fun!"
        'on enregistre la position du cadre de graphe PWP
        gauche = .Shapes(2).Left
        haut = .Shapes(2).Top
        longeur = .Shapes(2).Width
        hauteur = .Shapes(2).Height
        'on supprime la forme pour la remplacer par le graphe
        'se trouvant sur la première feuille Excel et ce aux mêmes
dimensions
        .Shapes(2).Delete
        shtExcel.Activate
        'ici on teste si le graphique à copier est en tant qu'objet
        'ou en pleine feuille avant de le copier
        If TypeName(shtExcel) = "Worksheet" Then
            'on copie alors le seul graphique disponible sur la première
feuille
            shtExcel.ChartObjects(1).Copy
        Else
            'on copie alors l'aire du graphique qui est en tant que feuille
entière
            shtExcel.ChartArea.Copy
        End If
        .Shapes.Paste
```

```
         'ou en tant qu'image Shapes.PasteSpecial (ppPasteMetafilePicture)
        .Shapes(2).Left = gauche
        .Shapes(2).Top = haut
        .Shapes(2).Width = longeur
        .Shapes(2).Height = hauteur
    End With

End Sub
```

17.3.3.2 Copie d'un graphique MS Excel dans diaporama MS PowerPoint existant

```
Sub AutomatePWP()
'Ne pas oublier d'ajouter la référence powerpoint

Set PPT = New PowerPoint.Application
With PPT
    .Visible = True
    .Activate
End With
My_Pres = "c:\test.ppt"

Set Pres = PPT.Presentations.Open(My_Pres)
Set Exl_Sh = Workbooks("Graph.xls").Sheets("Graphique")
Set My_Chart = Exl_Sh.ChartObjects(1)

PPT.Windows(1).View.GotoSlide 1

With Pres.Slides(1)
    .Shapes(1).TextFrame.Textrange.Text = "V.B.A. C'est fun!"
    Gauche = .Shapes(2).Left
    Haut = .Shapes(2).Top
    Longeur = .Shapes(2).Width
    Hauteur = .Shapes(2).Height
    .Shapes(2).Delete
    Exl_Sh.Activate
    'ici on teste si le graphique à copier est en tant qu'objet
    'ou en pleine feuille avant de le copier
    If TypeName(Exl_Sh) = "Worksheet" Then
        'on copie alors le seul graphique disponible sur la première
feuille
        'en tant qu'objet pour le coller en tant qu'objet XL dans
PowerPoint
        My_Chart.Copy
        'en tant qu'image pour le coller en tant qu'image dans PowerPoint
        'My_Chart.CopyPicture Appearance:=xlPrinter, Format:=xlPicture
    Else
        'on copie alors l'aire du graphique qui est en tant que feuille
entière
        Exl_Sh.ChartArea.Copy
    End If
    .Shapes.Paste
    'ou en tant qu'image Shapes.PasteSpecial (ppPasteMetafilePicture)
    .Shapes(2).Left = Gauche
    .Shapes(2).Top = Haut
    .Shapes(2).Width = Longeur
    .Shapes(2).Height = Hauteur
End With
End Sub
```

17.3.4 Automation MS Outlook

Avant de faire de l'automation MS Outlook, voyons d'abord simplement la commande permettant d'envoyer le classeur actif en tant que pièce jointe:

```
ActiveWorkbook.SendMail Recipients:="john@abc.com", Subject:="Test",
ReturnReceipt:=False
```

Un autre exemple d'automation dans lequel nous envoyons un mail à partir de variables définies en dur. L'exemple peut être complexifié à souhait:

```
'Cette routine envoie les paramètres à la sous-routine d'envoi de mail

Sub AutomateOutlook()

    'on pourrait utiliser cette variable attach pour envoyer en piece
    'jointe le classeur actif
    attach = ActiveWorkbook.Path & "\" & ActiveWorkbook.Name
    SendMail "isoz@microsoft.com", "Ici le sujet du message", "Ici le corps
du message"

End Sub

'C'est ici qu'on va instancier Outlook
'Si on veut se débarrasser des Security Warning de Outlook lors de son
lancement il faut signer la macro avec un certificat payant

Sub SendMail(email As String, ccmail as string, object As String, text As
String, Optional fileAttach As String)
    'A partir de Outlook 2007 il faut ajouter les lignes de code suivantes
pour ouvrir Outlook au préalable car sinon cela na marche pas
    'Set Otl=CreateObject("WScript.Shell")
    'Otl.Run "outlook.exe",1,False
    'Et pour vérifier si Outlook 2007 est déjà ouvert (sinon cela plante si
on l'ouvre deux fois) il suffit d'écrire la ligne suivante et de jouer avec
de la gestion d'erreurs pour palier aux IF qui ne gèrent pas les Set…
    'Set objOutlook = GetObject(, "Outlook.Application")
    Dim appOL As New Outlook.Application
    Set appOL = CreateObject("Outlook.Application")
    Dim mailOL As Outlook.mailitem
    Set mailOL = appOL.CreateItem(olMailItem)
    Dim Dest As String
    Dim Splitter() As String

    With mailOL
        .Recipients.Add email
        'Si la variable email contient plusieurs emails on est obligé de
faire la pirouette suivante:
        'Splitter = Split(email, ";")
        'For Each Dest In Splitter
            '.Recipients.Add (Trim(Dest))
        '.Next
        .Subject = object
        .Body = text
        If ccmail<>"" then
            With .Recipients.Add(ccmail)
                    .Type=olCC
            End With
        End If
```

```
        If fileAttach <> "" Then
            .Attachments.Add fileattach
        End If
    .send
    End With
    'Eventuellement si on veut fermer outlook…
    'Set appOL=Nothing
    'Set mailOL=Nothing
End Sub
```

ou pour lire le conteu du carnet d'adresse Outlook (il existe plsueiurs manière de faire cela):

```
Sub ImportOutl()
    Dim objOle As Outlook.Application
    Dim objNamespace As Namespace
    Dim objAddrList As AddressList
    Dim objAddrEntries As AddressEntries
    Dim objAdrEntry As AddressEntry
    Dim i As Integer

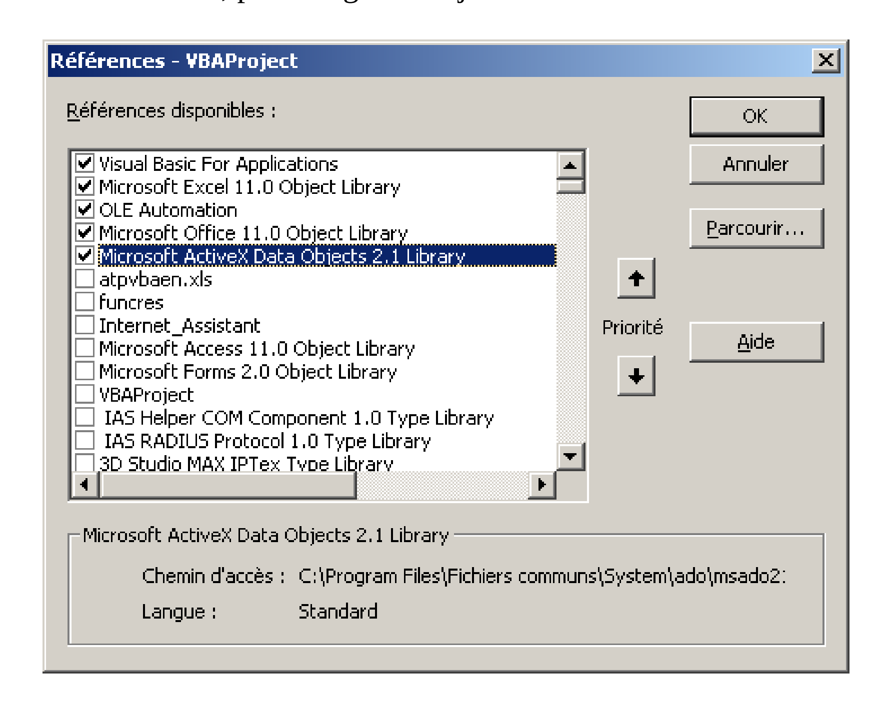
    Set objOle = CreateObject("Outlook.application")
    Set objNamespace = objOle.GetNamespace("MAPI")
    Set objAddrList = objNamespace.AddressLists("Contacts")
    Set objAddrEntries = objAddrList.AddressEntries
    Set objAdrEntry = objAddrEntries.GetFirst

    For i = 1 To objAddrEntries.Count
        Cells(i, 1) = objAdrEntry.Name
        Cells(i, 2) = objAdrEntry.Address
        Set objAdrEntry = objAddrEntries.GetNext
    Next i

End Sub
```


17.3.5 Automation MS Access

Pour l'automation MS Access, prendre garde à ajouter les bonnes références:



Dans l'exemple suivant, nous ouvrons une base de données *Magasin.mdb* et insérons dans la table *tblClients* un nouveau client à partir des informations se trouvant dans deux cellules MS Excel pour les mettre ensuite dans les champs *strName* et *strFname* de la table *tblClients*:

```
Sub InsererDonness()

    'Ne pas oublier d'ajouter la reference OLE Automation dans le projet
VBA!
    Dim ADOCnn As New ADODB.Connection, ADOTab As New ADODB.Recordset
    ADOCnn.Provider = "Microsoft.Jet.OLEDB.4.0"
    ADOCnn.Open ("c:/Magasin.mdb")
    ADOTab.Open "tblClients", ADOCnn, adOpenDynamic, adLockOptimistic

    If (Range("C6") = "" Or Range("C7") = "") Then
        MsgBox "Veuillez d'abord saisir le nom et le prénom du client!",
vbInformation
    Else
        ADOTab.AddNew
        ADOTab!strNom = Range("C7")
        ADOTab!strPrenom= Range("C6")
        ADOTab.Update
    End If
    ADOCnn.Close
    Set ADOCnn = Nothing

End Sub
```

Cet autre exemple écrit dans la première colonne MS Excel, les noms et prénoms de tous les clients de l'exemple précédent:


```
Sub LireDonnees()

    Dim ADOCnn As ADODB.Connection, ADOTab As ADODB.Recordset
    Dim Index As Long
    Set ADOCnn = New ADODB.Connection
    ADOCnn.Provider = "Microsoft.Jet.OLEDB.4.0"
    ADOCnn.Open ("c:/Magasin.mdb")
    Set ADOTab = New ADODB.Recordset
    ADOTab.Open "tblClients", ADOCnn, adOpenDynamic, adLockOptimistic

    Do While ADOTab.EOF = False
        Index = Index + 1
        Cells(Index, 1) = ADOTab!strName & " " & ADOTab!strFName
        ADOTab.MoveNext
    Loop
    ADOCnn.Close
    Set ADOCnn = Nothing

End Sub
```

Dans cet exemple, nous cherchons le nom d'un client saisi dans la cellule C2 de la feuille active pour renvoyer dans une cellule MS Excel son Nom plus son Prénom:

```
Sub CercheDonnee()

    Dim ADOCnn As New ADODB.Connection, ADOTab As New ADODB.Recordset
    ADOCnn.Provider = "Microsoft.Jet.OLEDB.4.0"
    ADOCnn.Open ("c:/Magasin.mdb")
    ADOTab.Open "tblClients", ADOCnn, adOpenDynamic, adLockOptimistic
```

```
    With ADOTab
        .Find "strName Like '*" & Range("C2") & "*'"
        If .EOF Then
            MsgBox "Aucun client de ce nom trouvé!", vbInformation
        Else
            Range("C2") = !strName & " " & !strFName
            'ou pour supprimer l'enregistrement plutôt que de l'afficher on
écrira:
            '.delete
            'Pour le modifier (mettre à jour) plutôt que de l'afficher on
écrira par exemple:
            'strName = Range("C2")
        End If
    End With
    ADOCnn.Close
    Set ADOCnn = Nothing

End Sub
```

L'exemple ci-dessous montre comment ajouter le contenu de deux cellules (le nom et le prénom d'un client en l'occurrence) dans un nouvel enregistrement de notre table *tblClients*:

```
Sub AjoutDonnees()

    Dim ADOCnn As New ADODB.Connection, ADOTab As New ADODB.Recordset
    ADOCnn.Provider = "Microsoft.Jet.OLEDB.4.0"
    ADOCnn.Open ("c:/Magasin.mdb")
    ADOTab.Open "tblClients", ADOCnn, adOpenDynamic, adLockOptimistic
    If (Range("C6") = "" Or Range("C7") = "") Then
        MsgBox "Veuillez saisir nom et prénom s.v.p!", vbInformation
    Else
        ADOTab.AddNew
        ADOTab!strNom = Range("C7")
        ADOTab!strPrenom = Range("C6")
        ADOTab.Update
    End If
    ADOCnn.Close
    Set ADOCnn = Nothing

End Sub
```


De même, pour mettre à jour le nom d'un client:

```
Sub MAJ()

    Dim ADOCnn As New ADODB.Connection, ADOTab As New ADODB.Recordset
    Dim Namen() As String, Zähler As Long
    ADOCnn.Provider = "Microsoft.Jet.OLEDB.4.0"
    ADOCnn.Open ("c:\Magasin.mdb")
    ADOTab.Open "tblClients", ADOCnn, adOpenStatic, adLockOptimistic

    ADOTab.MoveFirst
    ADOTab.Find "strNom Like '*" & Range("A1") & "*'"
    ADOTab!strNom = Range("B1")
    ADOTab.Update

    ADOCnn.Close
    Set ADOCnn = Nothing
```

```
End Sub
```

Passons maintenant à un élément qui nous intéresse tout particulièrement: les contrôles OLE ou ActiveX.

17.4 ActiveX

Les contrôles OLE, renommés ActiveX, sont des composants autonomes permettant de réaliser des applications. Ces composants s'appuient sur COM et OLE Automation que nous venons de voir.

Les contrôles OLE de première génération, appelés *contrôles OCX* (*OLE Control eXtension*), étaient des composants pouvant réagir à des événements extérieurs. La deuxième version de ces contrôles, désormais appelés *contrôles ActiveX* étend les possibilités d'OCX en permettant d'utiliser des composants distribués sur plusieurs systèmes interconnectés par un réseau tel qu'Internet, exploitant le modèle DCOM, extension de COM, que nous décrirons plus loin. Une application incorporant des contrôles ActiveX est appelée, nous l'avons déjà dit, un *document*. Notons qu'un même document peut contenir des contrôles de types très différents comme des tableaux Excel ou des documents HTML.

Une illustration de la puissance de ce mécanisme est par exemple l'affichage d'un document Word via MSIE. On constate en effet que, dès qu'un tel fichier est récupéré sur un serveur web, il y a incorporation de la barre d'outils de Word *dans* la fenêtre de MSIE, de façon transparente pour l'utilisateur. Ce mécanisme est supérieur à celui que nous avons vu dans le paragraphe précédent qui permettait de manipuler un composant incorporé dans un document, bien qu'un contrôle ActiveX puisse également faire l'objet d'une telle manipulation.

En résumé, un contrôle ActiveX peut être piloté par un document le comprenant, via une interface OLE Automation mais il peut également envoyer des ordres au document le contenant, via un deuxième type d'interface, comme nous venons de l'illustrer.

On distingue par ailleurs le *container ActiveX*, celui qui contient un contrôle, du *component ActiveX* qui est le composant en lui même. Concernant un container, il contient un certain nombre de propriétés de type *ambient* qui sont partagées entre tous les contrôles ActiveX d'un même document. Il peut s'agir par exemple de la couleur ou de la taille de la police de caractères, etc. L'accès à ces données se fait via une interface IDispatch que nous avons présentée plus haut.

Un container doit également pouvoir répondre aux événements générés par un composant ActiveX, également via une interface IDispatch.

Un composant ActiveX quant à lui est caractérisé par un certain nombre de propriétés et d'événements. On distingue quatre types de propriétés:

- ambiantes (*ambient*)
- standard
- étendues
- spécifiques

Les propriétés ambiantes sont celles que nous avons vues plus haut et sont partagées par tous les composants ActiveX d'un même document. En cas de changement dans ces propriétés, tous les composants sont prévenus par la modification d'un *flag* spécifique

Les propriétés standard sont quant à elles spécifiques à un composant ActiveX et sont similaires aux propriétés ambiantes (couleur, taille des caractères,...).

Les propriétés étendues sont associées à un composant mais ce n'est pas lui qui en a le contrôle. Elles peuvent permettre, par exemple, de spécifier un comportement par défaut dans une boite de dialogue. C'est le document container qui peut les manipuler.

Les propriétés spécifiques sont, comme leur nom l'indique, propres à un composant précis. Elles sont définies par le programmeur du composant.

Les événements, générés par un composant et traités par un container, sont classés en quatre catégories distingues permettant ainsi d'en faciliter l'accès au document:

- événements de type requête,
- événements générés *avant* une opération,
- événements générés *après*,
- événements générés *pendant* une opération.

Les requêtes sont envoyées par le composant à un document pour lui demander l'exécution d'une certaine action.

Les événements envoyés au document avant et après une opération sont reçus par le document, sans qu'il n'ait à faire une action particulière. On signale juste au document que le composant va débuter et a terminé une certaine opération.

Enfin, les événements produits durant une opération sont en fait des événements standard que l'on rencontre fréquemment dans les interfaces graphiques, à savoir un clic ou déplacement de souris, etc.

Il convient également de préciser que tout composant ActiveX doit être identifié auprès de la base des registres OLE afin qu'il puisse être utilisable. Cette opération est produite automatiquement lorsqu'on récupère un contrôle à partir d'une page web, via la balise HTML <OBJECT> dont nous avons parlé brièvement au chapitre précédent. D'ailleurs, ce contrôle est identifié de manière unique par un GUID, dont nous avons également déjà parlé.

Abordons maintenant le dernier service de l'architecture OLE que nous présenterons dans ce chapitre, à savoir l'extension DCOM du modèle COM.

17.5 DCOM Distributed (Distributed com)

Distributed COM est la version distribuée du modèle COM qui permet à des objets appartenant à différentes machines de communiquer entre eux via un réseau, alors que COM est limité à une même machine.

DCOM est basé sur l'environnement DCE (*Distributed Computing Environment*) , standardisé par l'OSF, dont il exploite en particulier les appels de procédures distantes RPC (*Remote Procedure Call*).

Par conséquent, à chaque fois qu'on envoie un message DCOM à un objet distant, ce message est en réalité transporté via RPC.

Dans un environnement DCOM, chaque machine impliquée dispose d'un serveur dit *Object Explorer* qui a pour but de gérer les objets COM disponibles sur la machine considérée.

Quand une application distante fait référence à un objet, l'Object Explorer vérifie que cet objet existe bien dans la base du système sur lequel il tourne. Si c'est le cas, l'accès à cet objet se fait via une évolution de l'interface IUnknown que nous avons présenté plus haut, appelée IRemUnknown pour *Remote Unknown*.

Précisons enfin que la gestion des références d'accès aux objets dans un environnement DCOM est quelque peu améliorée pour tenir compte des aléas d'une communication via un réseau. En effet, en cas de plantage d'une machine distante, les références enregistrées par un client ou un serveur risqueraient de ne plus être les mêmes. C'est pourquoi DCOM inclut un mécanisme de *ping* des clients vers les serveurs permettant ainsi d'indiquer à un serveur qu'un client est toujours opérationnel.

Notons par ailleurs que des requêtes ping (vers plusieurs objets d'un même serveur) peuvent être regroupées afin de ne pas sur charger un réseau.

18. SQL

Nous allons voir dans ce chapitre comment attaquer une base de données MS Access avec du SQL ainsi qu'un fichier MS Excel. Nous supposerons connu par le lecteur, les possibilités et le potentiel qu'offrente le SQL. Pour attaquer la base de données MS Access en V.B.A. il nous faudra utiliser les mêmes références qu'indiquées lors du chapitre sur l'automation.

18.1 Attaquer en SQL avec ADO un table MS Access

Dans l'exemple ci-dessous, nous allons voir comment faire une requête simple en SQL dans la table *tblClients* et retourner quelques informations dans une feuille MS Excel:

```
Sub PlainTextQuery()

    Dim rsData As ADODB.Recordset
    Dim szConnect As String
    Dim szSQL As String

    szConnect = "Provider=Microsoft.Jet.OLEDB.4.0; Data
Source=C:\Magasin.mdb;"
    'on peut faire aussi des requêtes d'analyses croisées si on le souhaite
ou inclure des variables V.B.A. dans la requête
    szSQL = "SELECT strNom, strPrenom FROM tblClients WHERE
strPays='Suisse' ORDER BY strNom"

    Set rsData = New ADODB.Recordset
    rsData.Open szSQL, szConnect, adOpenForwardOnly, adLockReadOnly,
adCmdText

    If Not rsData.EOF Then
        Sheet1.Range("A2").CopyFromRecordset rsData
        rsData.Close
        With Sheet1.Range("A1:B1")
            .Value = Array("Nom", "Prénom")
            .Font.Bold = True
        End With
        Sheet1.UsedRange.EntireColumn.AutoFit
    Else
        MsgBox "Error"
    End If

    If CBool(rsData.State And adstateopen) Then rsData.Close
    Set rsData = Nothing
End Sub
```


18.2 Attaquer en SQL avec ADO un fichier texte

Considérons maintenant le petit fichier texte ci-dessous (qui n'est qu'un export de notre table MS Access *tblClients*):

Clients.txt - Bloc-notes

Fichier Edition Format Affichage ?

```
"idClient";"olePhoto";"strPrenom";"strNom";"strRue";"intNbRue";"strNPA";"strCanton";"strPays";"bo
1;;"Alain";"Dutron";"Ch. de Chandieu";8;"1006";;"Suisse";1
2;;"Charlie";"Angel";"Arastrasse";20;"3048";;"France";0
3;;"Albert";"Boltzmann";"Ch. des Grottes";30;"1021";;"Suisse";0
```

Pour attaquer en SQL ce fichier texte avec les mêmes critères que l'exemple précédent, nous faisons usage d'un code quasi similaire:

```
Sub PlainTextQuery()

    Dim rsData As ADODB.Recordset
    Dim szConnect As String
    Dim szSQL As String

    szConnect = "Provider=Microsoft.Jet.OLEDB.4.0; Data Source=C:\;
Extended Properties=Text;"
    'on peut faire aussi des requêtes d'analyses croisées si on le souhaite
ou inclure des variables V.B.A. dans la requête
    szSQL = "SELECT strNom, strPrenom FROM Clients.txt WHERE
strPays='Suisse' ORDER BY strNom"

    Set rsData = New ADODB.Recordset
    rsData.Open szSQL, szConnect, adOpenForwardOnly, adLockReadOnly,
adCmdText

    If Not rsData.EOF Then
        Sheet1.Range("A2").CopyFromRecordset rsData
        rsData.Close
        With Sheet1.Range("A1:B1")
            .Value = Array("Nom", "Prénom")
            .Font.Bold = True
        End With
        Sheet1.UsedRange.EntireColumn.AutoFit
    Else
        MsgBox "Error"
    End If
    Set rsData = Nothing
End Sub
```


18.3 Attaquer en SQL avec ADO une requête MS Access

Voyons maintenant comment exécuter une requête *qryClients* se trouvant dans la base MS Access et en retourner le résultat (noms et prénoms de clients):

```
Sub CallQuery()

    Dim objField As ADODB.Field
    Dim rsData As ADODB.Recordset
    Dim loffset As Long
    Dim szconnect As String

    szconnect = "Provider=Microsoft.Jet.OLEDB.4.0; Data
Source=C:\Magasin.mdb;"
```

```
    Set rsData = New ADODB.Recordset
    rsData.Open "qryClients", szconnect, adOpenForwardOnly, adLockReadOnly,
adCmdTable

    'Pour être sûr que nous ayons la totalité du résultat en retour
    If Not rsData.EOF Then
        'On ajoute la ligne de titre à la feuille Excel
        With sheet1.Range("A1")
            For Each objField In rsData.Fields
                .Offset(0, loffset).Value = objField.Name
                loffset = loffset + 1
            Next objField
            .Resize(1, rsData.Fields.Count).Font.Bold = True
        End With
        'on pose le contenu du recordset dans la feuille
        sheet1.Range("A2").CopyFromRecordset rsData
        sheet1.UsedRange.EntireColumn.AutoFit
    Else
        MsgBox "Erreur: aucun enregistrement retourné"
    End If

    rsData.Close
    Set rsData = Nothing

End Sub
```

18.4 Attaquer en SQL avec ADO une feuille MS Excel

Considérons maintenant le fichier *Commandes.xls* enregistré sur *C:* avec le contenu suivant:

Commandes.xls

	A	B	C	D	E	F	G
1	N° Client	Secteur d'activité	N° Commande	Date Commande	Article	Nombre	Prix par pièce
2	100	Alimentaire	1	03.01.2000	Compaq Pres	12	1650
3	123	Machines/Outils	2	03.01.2000	IBM 500	2	2299
4	109	Assurances	3	03.01.2000	AST Intel 150	5	2690
5	104	Assurances	4	03.01.2000	AST Intel 200	3	3190
6	117	Banques	5	04.01.2000	Compaq Pres	13	1650
7	103	Alimentaire	6	04.01.2000	AST Intel 150	2	2690
8	104	Assurances	7	04.01.2000	AST Intel 200	2	3190
9	111	Education	8	04.01.2000	IBM 500	4	2299
10	113	Construction	9	04.01.2000	Compaq Pres	4	1650
11	116	Pharmaceutique	10	04.01.2000	IBM 500	2	2299
12	110	Distribution	11	05.01.2000	AST Intel 200	6	3190
13	112	Machines/Outils	12	05.01.2000	Compaq Pres	6	1650
14	123	Machines/Outils	13	05.01.2000	IBM 500	6	2299
15	113	Construction	14	05.01.2000	AST Intel 150	3	2690
16	115	Distribution	15	05.01.2000	Compaq Pres	8	1650
17	124	Assurances	16	05.01.2000	AST Intel 200	8	3190
18	124	Assurances	17	05.01.2000	Compaq Pres	11	1650
19	106	Construction	18	05.01.2000	AST Intel 200	11	3190
20	101	Construction	19	05.01.2000	Compaq Pres	14	1650
21	116	Pharmaceutique	20	06.01.2000	IBM 500	7	2299
22	112	Machines/Outils	21	06.01.2000	AST Intel 150	6	2690
23	125	Construction	22	06.01.2000	Compaq Pres	23	1650
24	100	Alimentaire	23	06.01.2000	IBM 500	3	2299
25	125	Construction	24	06.01.2000	AST Intel 200	2	3190

Commandes

Attaquons ce fichier en SQL:

```
Sub QueryWorksheet()
```

```
    Dim rsData As ADODB.Recordset
    Dim szConnect As String
    Dim szSQL As String

    szConnect = "Provider=Microsoft.jet.oledb.4.0; Data
Source=C:\Commandes.xls; Extended Properties=Excel 8.0;"

    'si la première ligne ne contient pas d'étiquettes (car le moteur Jet
de OLE DB en attend
     une) alors il faut écrire:
    'szConnect = "Provider=Microsoft.jet.oledb.4.0; Data
Source=C:\Commandes.xls;
     Extended Properties=Excel 8.0; HDR=No"

    'en utilisant le nom de la feuille
    szSQL = "SELECT * FROM [Commandes$] ORDER BY [Nombre]"

    'ou en utilisant une zone de cellules
    szSQL = "SELECT * FROM [Commandes$A1:F30]"

    'un peu plus compliqué
    'szSQL = "SELECT * FROM [Commandes$] WHERE [Article]='IBM 500' ORDER BY
[Nombre]"

    Set rsData = New ADODB.Recordset
    rsData.Open szSQL, szConnect, adOpenForwardOnly, adLockReadOnly,
adCmdText

    Sheet1.Range("A1").CopyFromRecordset rsData

    rsData.Close
    Set rsData = Nothing

End Sub
```


De meme, nous pouvons executer des requêtes SQL habituelles:

```
Sub QueryWorksheet()

    Dim rsData As ADODB.Recordset
    Dim szConnect As String
    Dim szSQL As String

    szConnect = "Provider=Microsoft.jet.oledb.4.0; Data
Source=C:\Commandes.xls; Extended Properties=Excel 8.0;"

    'en utilisant le nom de la feuille
    szSQL = "INSERT INTO [Commandes$]
VALUES('100','Alimentaire','102','01.01.2001','IBM
500','10','1300','1','21000','Non','DHL')"

    Set objConn = New ADODB.Connection
    objConn.Open szConnect

    objConn.Execute szSQL, , adCmdText Or adExecuteNoRecords
    objConn.Close

    Set objConn = Nothing

End Sub
```

18.5 Attaquer en SQL avec ADO une base Oracle

```
Sub Load_dataOracle()
    Dim cn As ADODB.Connection
    Dim rs As ADODB.Recordset
    Dim col As Integer
    Dim row As Integer
    Dim Query As String
    Set cn = New ADODB.Connection
    Set rs = New ADODB.Recordset

    Query = "select * from employees where employee_id = 100"

    cn.Open "User ID= " & UserForm.txtusrname & ";Password=" &
UserForm.txtPassword & ";Data Source=" & UserForm.cboInstance &
";Provider=MSDAORA.1"

    rs.Open Query, cn

    col = 0
     'First Row: names of columns
    Do While col < rs.Fields.Count
        Sheet1.Cells(1, col + 1) = rs.Fields(col).Name
        col = col + 1
    Loop

    'Now actual data as fetched from select statement
    row = 1
    Do While Not rs.EOF
        row = row + 1
        col = 0

        Do While col < rs.Fields.Count
            Sheet1.Cells(row, col + 1) = rs.Fields(col).Value
            col = col + 1
        Loop

        rs.MoveNext
    Loop
     rs.Close
     Set rs = Nothing
End Sub
```

18.6 Attaquer en SQL avec ADO une base SQL Server

```
Sub UpdateTable()
   Dim rngName As Range
   cnnstr = "Provider=SQLOLEDB; " & _
            "Data Source=MyServer; " & _
            "Initial Catalog=Mydb;" & _
            "User ID=User;" & _
            "Password=Pwd;" & _
            "Trusted_Connection=No"
   Set rngName = ActiveCell
   'Debug.Print (rngName)
   Set cnn = New ADODB.Connection
   Application.ScreenUpdating = False
   cnn.Open cnnstr
   Set rs = New ADODB.Recordset
   uSQL = "UPDATE MyTable SET FieldNameX = 1 WHERE FieldNameY = '" &
rngName & "' "
   rs.CursorLocation = adUseClient
   rs.Open uSQL, cnn, adOpenStatic, adLockOptimistic, adCmdText
   rs.Close
   Set rs = Nothing
   cnn.Close
   Set cnn = Nothing
End Sub
```

19. DDE (Dynamic Data Exchange)

DDE est un protocole de communication créé par Microsoft pour permettre aux applications de l'environnement Windows l'envoi ou la réception de données ainsi que l'échange d'instructions entre elles.

Il instaure une relation client/serveur entre les deux applications. L'application serveur fournit les données et accepte de répondre aux demandes d'informations de toute application intéressée par ses données. Les applications qui font les demandes sont appelées "clients". Certaines applications comme MS Excel peuvent être à la fois client et serveur.

Pour obtenir les données d'une autre application, le programme client ouvre un canal vers l'application serveur en spécifiant trois choses:

- Nom de l'application
- Nom du sujet ou **TOPIC**
- Nom de la donnée ou **ITEM**

Par exemple, dans le cas d'Excel, le nom de l´application est "EXCEL", le *TOPIC* est le nom de la feuille de calcul qui contient les données et l'*ITEM* représente les coordonnées d'une cellule de cette feuille de calcul.

Quand une application client a établi un lien avec une application serveur DDE, elle envoie des requêtes d´abonnement ("advises") aux Items du serveur. Le serveur devra aviser le client quand la valeur d´un des Items aura changé. Ce lien restera actif tant que le client ou le serveur resteront en connexion. C'est un mécanisme d'échange de données efficace car événementiel.

Voici un exemple de DDE entre Excel et Word:

```
'*******************************
'Créateur: Vincent ISOZ
'Dernière modification: 20.11.2003
'Nom procédure: DDEWord()
'Commentaires: quelques petits exemples sympas avec le bloc-notes
'*******************************

Sub DDEWord()


    Application.DDEInitiate app:="WinWord", topic:="c:\test.doc"
…'on va envoyer un ordre à Word pour coller la cellule qui a été copiée
dans Excel au préalable
    Set Datarange = ThisWorkbook.Worksheets("Feuil").Range("a1")
    Datarange.Copy
    Application.DDEExecute chan, "[EDITPASTE]"
    'on envoie un ordre d'impression via DDE à word
    Application.DDEExecute channelNumber, "[FILEPRINT]"
    Application.DDETerminate channelNumber

    Application.DDETerminate chan

End Sub
```

20. API ET DLL

Les API (Application Program Interface) sont des appels à des routines qui sont contenues dans les DLL (Dynamic Linked Library).

Voici la liste des fichiers DLL les plus usités mais nous verrons dans ce document seulement ceux que l'auteur connaît (…).

Advapi32.DLL	Appels de sécurité, appels de service et registre
comdlg32.dll	Boîtes de dialogues communes (ouvrir, enregistrer, ...)
Gdi32.DLL	Fonctions graphiques
Kernel32.DLL	Noyau Windows (Mémoire, disque, CPU, ...)
LZ32.dll	Compression 32 bits
Mpr.DLL	Routeurs à fournisseurs multiples
netapi32.dll	Réseaux 32 bits
shell32.dll	API shell 32 bits
user32.dll	Interfaces utilisateurs (fenêtres, menus, ...)
version.dll	Gestion des différentes versions
Winmm.DLL	Multimedia (son, midi, ...)
winlnet.dll	Serveurs internet

Remarque: attention les DLL's sont Case-sensitives (car C++ derrière…)

20.1 Obtenir temps d'ouverture de session

La fonction API GetTickCount de kernel32 permet de connaître le temps en millisecondes depuis que Windows (ou l'ordinateur) est allumé. Comme cette valeur change toutes les millisecondes, il faut bien sûr l'appeler plusieurs fois, la stocker dans des variables temporaires, etc...

Il faut déclarer la fonction suivante en dehors de toute procédure au début d'un module:

```
Declare Function GetTickCount Lib "kernel32" () As Long
```

Ensuite, pour l'utiliser, il suffit d'appeler le nom de la fonction dans une procédure tel que par exemple:

```
Sub AfficheTemps()

    Dim intHeures As Integer
    Dim intMinutes As Integer
    Dim intSecondes As Double

    MsgBox GetTickCount / 3600000

    intTemps = GetTickCount
    intHeures = Int(intTemps / 3600000)
    intMinutes = Int((intTemps / 3600000 - intHeures) * 60)
    intSecondes = Int((intTemps / 3600000 - intHeures - intMinutes / 60) *
3600)
```

```
    MsgBox intHeures & ":" & intMinutes & ":" & intSecondes

End Sub
```

20.2 Récuperer nom ordinateur

La fonction API GetComputer peut s'avérer très utile pour tout développeur. A nouveau, le principe d'utilisation est similaire que dans l'exemple précédent, nous déclarons la fonction:

```
Declare Function RecupNomOrdinateur Lib "Kernel32.dll" Alias
"GetComputerNameA" (ByVal lpbuffer As String, nsize As Long) As Long
```

et pour l'utiliser nous écrivons dans un procédure un code du type:

```
Sub AfficheNomPC()

    Dim NomOrdinateur As String
    Dim Resultat As Long
    NomOrdinateur = String$(255, 32)
    Resultat = RecupNomOrdinateur(NomOrdinateur, 255)
    MsgBox NomOrdinateur

End Sub
```

20.3 Récuperer nom utilisateur MS Windows

Un autre outil puissant est la récupération du nom d'utilisateur à l'aide de l'API ADV afin de poser des droits d'accès aux fonctionnalités de votre code.

La méthode simple:

```
MsgBox vba.interaction.environ("USERNEMA")
```

ou la méthode compliquée utilisant la déclaration de la fonction est en tout point similaire aux techniques précédentes:

```
Declare Function RecupNomUtilisateur Lib "advapi32.dll" Alias
"GetUserNameA" (ByVal lpBuffer As String, nSize As Long) As Long
```

et l'implémentation du code étant du type (nous choisissons arbitrairement l'implémentation dans une fonction):

```
Function NomUtilisateur() As String

    Dim StrNomUtilisateur As String
    Dim Resultat As Long ' Contiendra simplement 1 si l'appel s'est bien
déroulé

    StrNomUtilisateur = String$(255, 0)
    Resultat = RecupNomUtilisateur(StrNomUtilisateur, 255)

    If Resultat = 1 Then
        NomUtilisateur = StrNomUtilisateur
    Else
```

```
        NomUtilisateur = "UTILISATEUR INCONNU"
    End If

End Function
```

et pour appeler la fonction et afficher le résultat à l'écran:

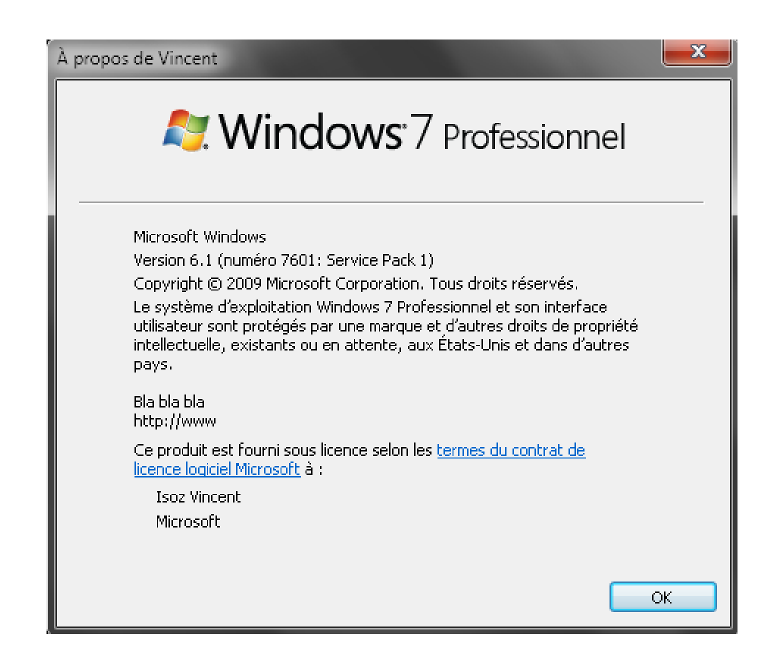
```
Sub Test()
    MsgBox NomUtilisateur
End Sub
```

20.4 Détecter si connextion Internet disponible

20.5 Afficher les informations systèmes de MS Windows

```
Private Declare Function ShellAbout Lib "shell32.dll" Alias "ShellAboutA"
(ByVal hwnd As Long, ByVal szApp As String, ByVal szOtherStuff As String,
ByVal hIcon As Long) As Long

Sub AfficheInfos()
   Dim hwnd As Long
   Dim Symbol As Long
   ShellAbout hwnd, "Vincent", "Bla bla bla" & vbCrLf & "http://www",
Symbol
End Sub
```



20.6 Détecter si connexion Internet disponible

20.7 Créer un Fichier Zip

```
Public Declare Sub Sleep Lib "kernel32" (ByVal dwMilliseconds As Long)

Sub DateiZippen()
'Early binding, set reference to: Microsoft Shell Controls and automation
(C:\WINNT\systems32\SHELL32.dll)
    NameDatei = "C:\Christmas Elements.psd"
    NameZipDatei = "C:\Example.zip"
```

```
    Dim oapp As Shell
    Set oapp = CreateObject("Shell.Application")

    'NewZip (NameZipDatei)
    If Len(Dir(NameZipDatei)) > 0 Then Kill NameZipDatei
    Open NameZipDatei For Output As #1
    Print #1, Chr$(80) & Chr$(75) & Chr$(5) & Chr$(6) & String(18, 0)
    Close #1
    oapp.Namespace(NameZipDatei).CopyHere NameDatei
    Do Until oapp.Namespace(NameZipDatei).items.Count = 1
        Sleep 1000
    Loop
End Sub
```

20.8 Afficher la structure arborescante de l'ordinateur

```
Type BROWSEINFO
    hOwner As Long
    pidlRoot As Long
    pszDisplayName As String
    lpszTitle As String
    ulFlags As Long
    lpfn As Long
    lParam As Long
    iImage As Long
End Type

Declare Function SHGetPathFromIDList Lib "shell32.dll" _
    Alias "SHGetPathFromIDListA" (ByVal pidl As Long, ByVal pszPath As _
String) As Long

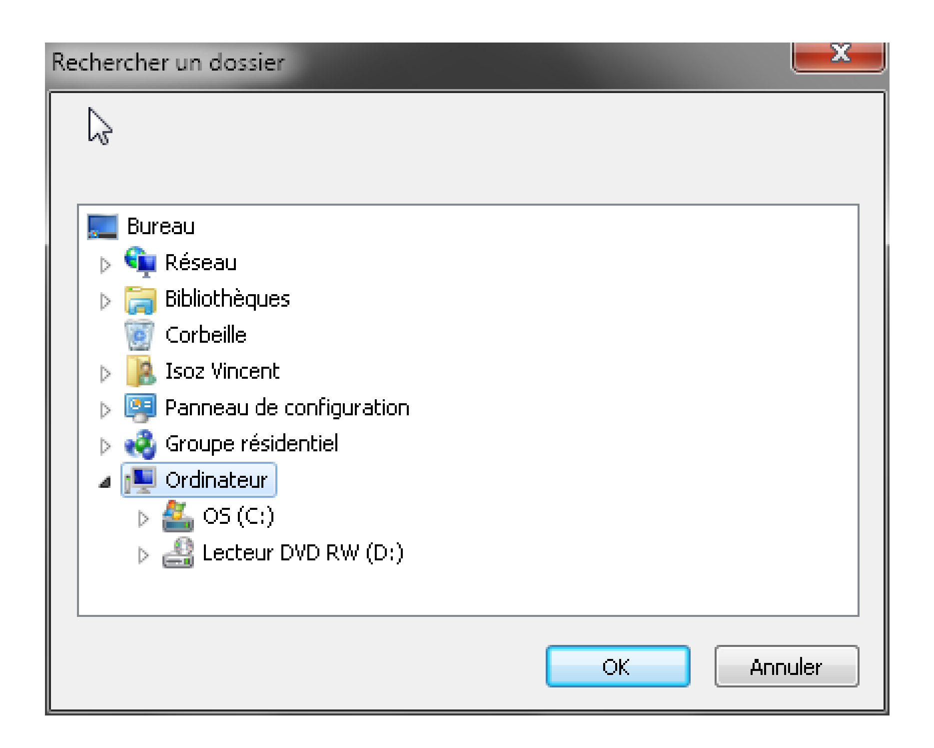
Declare Function SHBrowseForFolder Lib "shell32.dll" _
    Alias "SHBrowseForFolderA" (lpBrowseInfo As BROWSEINFO) As Long

Declare Function FindWindow Lib "user32" Alias "FindWindowA" (ByVal _
lpClassName As String, ByVal lpWindowName As String) As Long


Function SelectedPath(Msg) As String
Dim bInfo As BROWSEINFO
Dim path As String
Dim L As Long
    bInfo.pidlRoot = 0&
    L = SHBrowseForFolder(bInfo)
    path = Space$(512)
    If SHGetPathFromIDList(ByVal L, ByVal path) Then
        SelectedPath = Left(path, InStr(path, Chr$(0)) - 1)
    Else: SelectedPath = ""
    End If
End Function


Sub TreeStructure()
On Error Resume Next
Dim sPath As String
    sPath = SelectedPath(s)
    If sPath = "" Then Exit Sub
    MsgBox "Le dossier choisi est: " & sPath
End Sub
```


Ce qui donne en exécutant TreeStructure:



20.9 Vider le presse papier

Certaines personnes vous demanderont de créer une macro qui met des cellules ou des objets dans le presse-papier ce qui est simple puisqu'il suffit de faire un macro qui copie. Ce qui est moins trivial par contre c'est de vider le presse-papier.


```
Private Declare Function OpenClipboard Lib "user32" (ByVal hwnd As Long) As
Long

Private Declare Function EmptyClipboard Lib "user32" () As Long

Private Declare Function CloseClipboard Lib "user32" () As Long

Sub VidePressePapier()
    If OpenClipboard(0&) <> 0 Then
        Call EmptyClipboard
        Call CloseClipboard
    End If
End Sub
```

20.10 Jouer un son

Un classique pour ceux qui recréent des systèmes de rappels dans Microsoft Excel comme le fait Microsoft Outlook.

```
Declare Function sndPlaySound32 Lib "winmm.dll" Alias _
    "sndPlaySoundA" (ByVal lpszSoundName As String, ByVal uFlags As Long)
As Long

Sub JouerSon()
      Call sndPlaySound32("c:\chimes.wav", 0)
End Sub
```

et pour les ordinateurs n'ayant pas de carte son il faudra faire biper le son système:

```
Declare Function Beep Lib "kernel32.dll" (ByVal dwFreq As Long, ByVal
dwDuration As Long) As Long

Sub BeepOrdinateur()

    Dim L As Long
    L = Beep(250, 1000)

    If L = 0 Then
        MsgBox "L'ordinateur ne peut pas biper"
    End If

End Sub
```

21. APPLICATION EXTERNES

21.1 Lotus Notes

Pour ceux qui ont connu Lotus Notes, voici le code pour envoyer des courriels:

Considérons une feuille Microsoft Excel avec une colonne A contenant parfois le mot *Overdue* pour signaler des tâches en retard et une colonne B contenant l'adresse électronique des personnes devant travailler sur les tâches en retard. Voici le code qui envoie automatiquement un courriel à tout ce beau monde:

```
Sub Mailing_LotusNotes_PlainText()

    Dim x As Integer
    Dim UserName As String
    Dim MailDbName As String
    Dim Recipient As Variant
    Dim Maildb As Object
    Dim MailDoc As Object
    Dim Attachement As String
    Dim AttachME As Object
    Dim Session As Object
    Dim stSignature As String

    With Application
        .ScreenUpdating = False
        .DisplayAlerts = False
        ' Open and locate current LOTUS NOTES User
        For x = 2 To Cells(Rows.Count, "A").End(xlUp).Row
            If Range("A" & x) = "Overdue" Then
                Set Session = CreateObject("Notes.NotesSession")
                UserName = Session.UserName
                MailDbName = Left$(UserName, 1) & Right$(UserName,
(Len(UserName) - InStr(1, UserName, " "))) & ".nsf"
                Set Maildb = Session.GetDatabase("", MailDbName)
                    If Maildb.IsOpen = True Then
                         'Already open for mail
                    Else
                        Maildb.OPENMAIL
                    End If
                'Create New Mail and Address Title Handlers
                Set MailDoc = Maildb.CREATEDOCUMENT
                MailDoc.Form = "Memo"
                stSignature =
Maildb.GetProfileDocument("CalendarProfile").GetItemValue("Signature")(0)
                'Select range of e-mail addresses
                Recipient = Worksheets("Sheet1").Range("B" & x).Value
                MailDoc.SendTo = Recipient
                'MailDoc.SendFrom = Sender
                'MailDoc.CopyTo = ccRecipient
                'MailDoc.BlindCopyTo = bccRecipient
                MailDoc.Subject = "URGENT NOTIFICATION"
                MailDoc.Body = "Please ensure you update your files." &
vbCrLf & vbCrLf & stSignature
                'Set up the embedded object (Rich Text file) and attachment
and attach it
                If Attachment <> "" Then
```

```
                    Set AttachME = MailDoc.CREATERICHTEXTITEM("Attachment")
                    Set EmbedObj = AttachME.EMBEDOBJECT(1454, "",
Attachment, "Attachment")
                    MailDoc.CREATERICHTEXTITEM ("Attachment")
                End If
                MailDoc.SaveMessageOnSend = True
                MailDoc.PostedDate = Now()
            On Error GoTo errorhandler
                MailDoc.SEND 0, Recipient
                Set Maildb = Nothing
                Set MailDoc = Nothing
                Set Session = Nothing
                .ScreenUpdating = True
                .DisplayAlerts = True

errorhandler:
                Set Maildb = Nothing
                Set MailDoc = Nothing
                Set Session = Nothing
            End If
        Next x
    End With
End Sub
```

Et maintenant une version pour ceux qui préfèrent avoir un courriel en HTML:

```
Sub Mailing_LotusNotes_HTML()

    Dim richStyle As Object
    Dim richText As Object

    Set session = CreateObject("Notes.NotesSession")
    UserName = session.UserName
    MaildbName = Left$(UserName, 1) & Right$(UserName, (Len(UserName) -
InStr(1, UserName, ""))) & ".nsf"
        Debug.Print MaildbName
    Set Maildb = session.GetDatabase("", MaildbName)
    If Maildb.IsOpen = True Then
        Debug.Print "Lotus is already open"
    Else
        Maildb.OPENMAIL
        Debug.Print "Lotus has has been opened"
    End If

    bodyHTML = "< p>The following date is bold < b>21/12/2010</b>.  This
sentence is plain text.</p>" & _
        "< p>This new paragraph is plain text with a link <a
href=""www.google.com"">google</a> .</p>"

    'CHANGE REQUIRED - REMOVE THE SPACES IN EACH TAG

    HTML = "< HTML>" & vbLf & "< HEAD>" & vbLf & _
        "< META http-equiv=""Content-Type"" content=""text/html;
charset=ISO-8859-1"">" & vbLf & _
        "</HEAD>" & vbLf & _
        "< BODY>" & bodyHTML & "</BODY>" & vbLf & "</HTML>"

    session.ConvertMime = False     'Do not convert MIME to rich text

    For i = 2 To 1000 Step 1
        If Cells(i, 5).Value = "" Then
```

```
            Exit For
        End If
        Set MailDoc = Maildb.CreateDocument
        'To format with bold later the mail content
        Set NMIMEBody = MailDoc.CreateMIMEEntity
        Set richStyle = session.CreateRichTextStyle
        MailDoc.Form = "Memo"
        MailDoc.SendTo = Cells(i, 5).Value
        MailDoc.Principal = "bla_bla@domain.com"
        MailDoc.Subject = "Lettre de confirmation nouvelle classification
interne / Confirmation letter New Internal classification"

        Set NStream = session.CreateStream
        NStream.WriteText HTML
        NMIMEBody.SetContentFromText NStream, "text/html; charset=ISO-8859-
1", ENC_NONE
        MailDoc.SaveMessageOnSend = True
        MailDoc.PostedDate = Now()
        MailDoc.Send 0, Recipient
        Set MailDoc = Nothing
    Next i

    MsgBox "Job Done! Thanks for your patience.", vbOKOnly + vbInformation
End Sub
```

21.2 PDF (listing)

Voici une routine avec une fonction utilisant du VBScript qui va chercher tous les PDF contenus dans un dossier, va le lister dans la feuille avec le nombre de pages correspondant.


```
Sub Test()
   Dim MyPath As String, MyFile As String
   Dim i As Long
   MyPath = "C:\TestFolder"
   MyFile = Dir(MyPath & Application.PathSeparator & "*.pdf", vbDirectory)
   'on peut rajouter "+ vbReadOnly + vbHidden + vbSystem" après vbDirectory

   Range("A:B").ClearContents
   Range("A1") = "File Name": Range("B1") = "Pages"
   Range("A1:B1").Font.Bold = True
   i = 1
   Do While MyFile <> ""
      i = i + 1
      Cells(i, 1) = MyFile
      Cells(i, 2) = GetPageNum(MyPath & Application.PathSeparator & MyFile)
      MyFile = Dir
   Loop
   Columns("A:B").AutoFit
   MsgBox "Total of " & i - 1 & " PDF files have been found" & vbCrLf _
   & " File names and corresponding count of pages have been written on " _
   & ActiveSheet.Name, vbInformation, "Report..."
End Sub

Function GetPageNum(PDF_File As String)
   'Haluk 19/10/2008
   Dim FileNum As Long
   Dim strRetVal As String
   Dim RegExp
   Set RegExp = CreateObject("VBscript.RegExp")
```

```
    RegExp.Global = True
    RegExp.Pattern = "/Type\s*/Page[^s]"
    FileNum = FreeFile
    Open PDF_File For Binary As #FileNum
    strRetVal = Space(LOF(FileNum))
    Get #FileNum, , strRetVal
    Close #FileNum
    GetPageNum = RegExp.Execute(strRetVal).Count
End Function
```

21.3 PDF (lire les champs)

Considérons un PDF contenant deux simples champs nommés Text1 et Text2 et voici le code pour lire les PDF (ne pas oublier d'ajouter la référence Acrobat!):

```
Private Sub ReadPDFFormContent()
    Dim AcroApp As Acrobat.CAcroApp
    Dim theForm As Acrobat.CAcroPDDoc
    Dim jso As Object
    Dim text1, text2 As String

    Set AcroApp = CreateObject("AcroExch.App")
    Set theForm = CreateObject("AcroExch.PDDoc")
    theForm.Open ("C:\tmp\Form.pdf")
    Set jso = theForm.GetJSObject

    'get the information from the form fields Text1 and Text2
    text1 = jso.getField("Text1").Value
    text2 = jso.getField("Text2").Value

    MsgBox "Values read from PDF: " & text1 & " " & text2

    'set a text field
    Dim field2 As Object
    Set field2 = jso.getField("Text2")

    theForm.Close

    AcroApp.Exit
    Set AcroApp = Nothing
    Set theForm = Nothing

End Sub
```


21.4 PDF (écrire dans des champs)

Considérons un simple PDF avec un champ nommé *Text1*, nous voulons depuis V.B.A. écrire dans des champs (ne pas oublier d'ajouter la référence Acrobat!):

```
Sub WritePDFForm()
    Dim FileNm, gApp, avDoc, pdDoc, jso

    FileNm = "c:\tmp\Form.pdf" 'File location
    Set gApp = CreateObject("AcroExch.app")

    Set avDoc = CreateObject("AcroExch.AVDoc")
    If avDoc.Open(FileNm, "") Then
        Set pdDoc = avDoc.GetPDDoc()
```

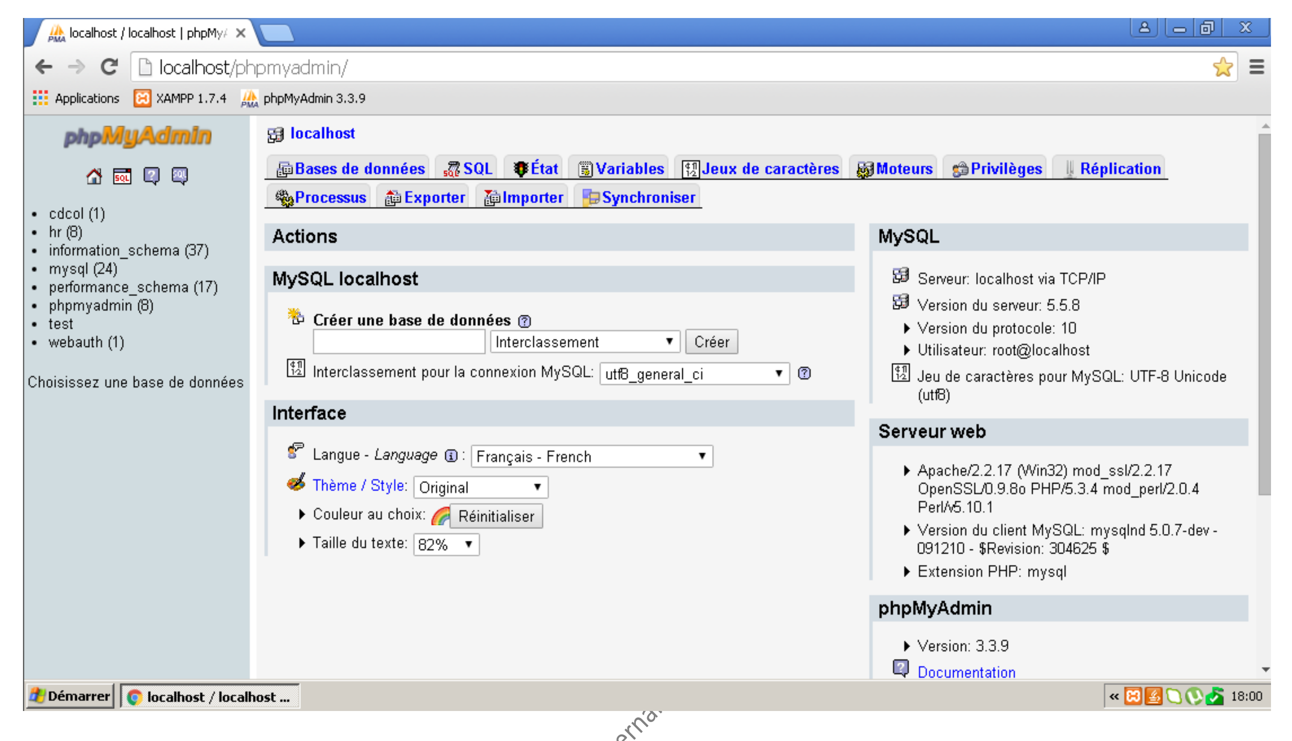
```
        Set jso = pdDoc.GetJSObject
        jso.getField("Text2").Value = "myValue"
        pdDoc.Save PDSaveIncremental, FileNm
        'Save changes to the PDF document
        pdDoc.Close
    End If

    'Close the PDF; the True parameter prevents the Save As dialog from
showing
    avDoc.Close (True)

    'Some cleaning
    Set gApp = Nothing
    Set avDoc = Nothing
    Set pdDoc = Nothing
    Set jso = Nothing
End Sub
```

21.5 Lire mySQL

Considérons une installation standard de MySQL dans un environnement Microsoft Windows:



Avec la base *HR* et la table *employees* ci-dessous:

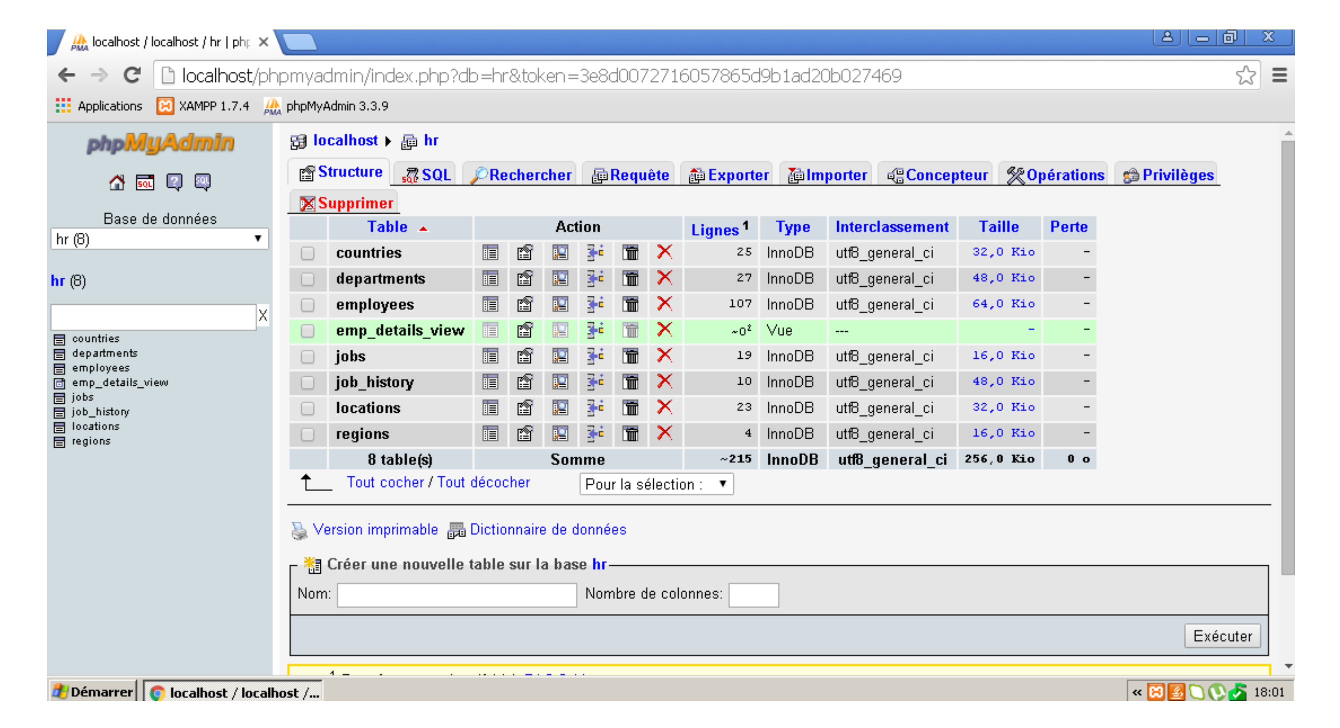
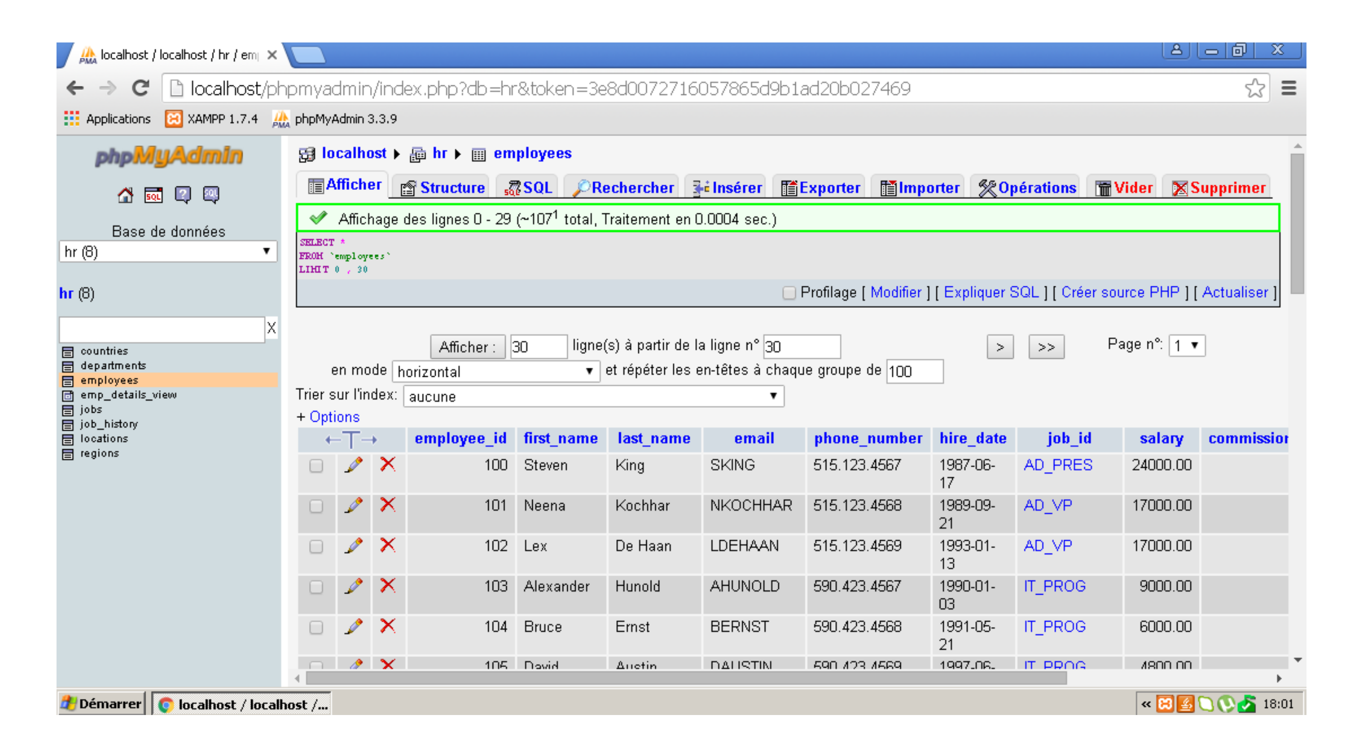


Table	Action	Lignes	Type	Interclassement	Taille	Perte
countries		25	InnoDB	utf8_general_ci	32,0 Kio	-
departments		27	InnoDB	utf8_general_ci	48,0 Kio	-
employees		107	InnoDB	utf8_general_ci	64,0 Kio	-
emp_details_view		~0	Vue	---	-	-
jobs		19	InnoDB	utf8_general_ci	16,0 Kio	-
job_history		10	InnoDB	utf8_general_ci	48,0 Kio	-
locations		23	InnoDB	utf8_general_ci	32,0 Kio	-
regions		4	InnoDB	utf8_general_ci	16,0 Kio	-
8 table(s)	Somme	~215	InnoDB	utf8_general_ci	256,0 Kio	0 o

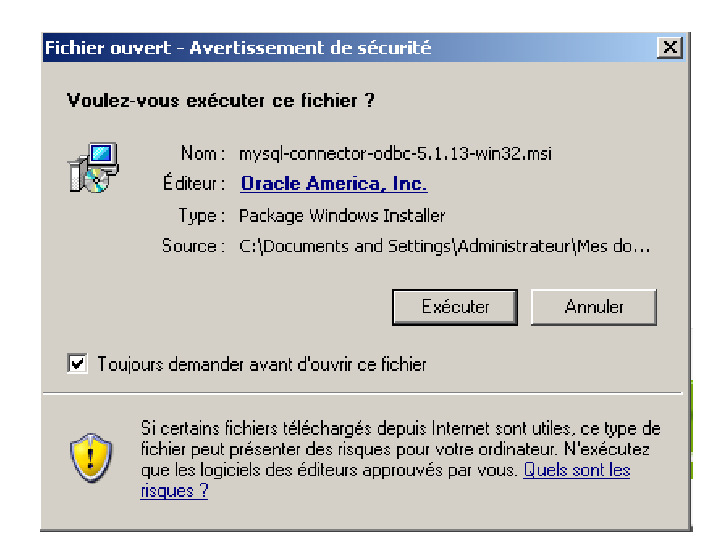
Nous avons:



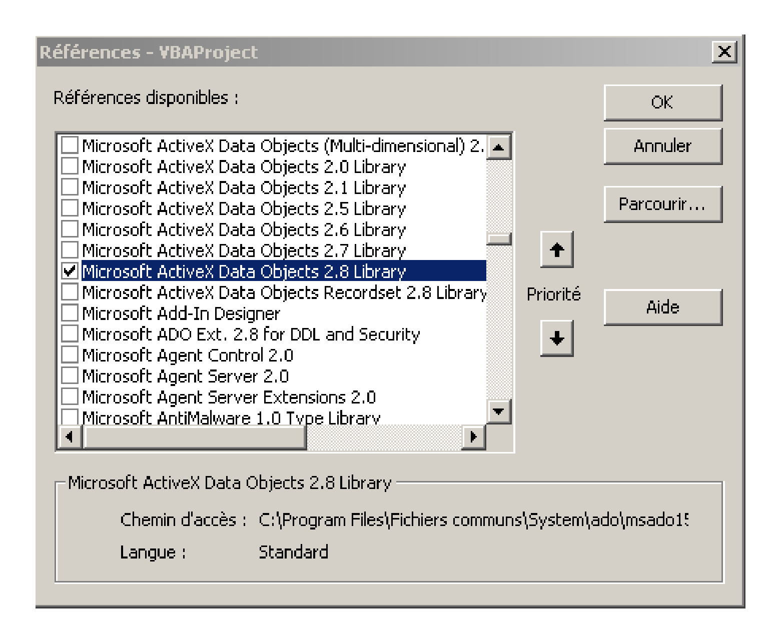
Ensuite on installe le connecteur ODBC de mySQL:

localhost / localhost / hr / em | MySQL :: Download Connect
https://dev.mysql.com/downloads/connector/odbc/5.1.html
Applications XAMPP 1.7.4 phpMyAdmin 3.3.9
Enterprise Community Yum Repository APT Repository SUSE Repository Windows Archives
MySQL on Windows
MySQL Yum Repository
MySQL APT Repository
MySQL SUSE Repository
MySQL Community Server
MySQL Cluster
MySQL Fabric
MySQL Router
MySQL Utilities
MySQL Workbench
MySQL Connectors
Connector/ODBC
Connector/Net
Connector/J
Connector/Python
Download Connector/ODBC
Connector/ODBC is a standardized database driver for Windows, Linux, Mac OS X, and Unix platforms
Online Documentation:
• MySQL Connector/ODBC Installation Instructions, Documentation and Change History
MySQL open source software is provided under the GPL License.
OEMs, ISVs and VARs can purchase commercial licenses.
Please report any bugs or inconsistencies you observe to our Bugs Database.
Thank you for your support!
Generally Available (GA) Releases
Connector/ODBC 5.1.13
Select Version: 5.1.13
Looking for the latest GA version?
Select Platform: Microsoft Windows
bugs.mysql.com
Démarrer MySQL :: Download Co... Microsoft Excel - Classeur1 Microsoft Visual Basic - Cl...
18:09

Ce qui donne:



Après avoir faire plusieurs **Next**… **Next**… dans VBA on ajoute la référence suivante:



Et le code correspondant:

```
Sub ExtractDataFromMySQL()

    Dim Password As String
    Dim SQLStr As String
    Dim Cn As ADODB.Connection
    Dim Server_Name As String
    Dim User_ID As String
    Dim Database_Name As String
    Dim rs As ADODB.Recordset
    Set rs = New ADODB.Recordset

    Server_Name = "localhost"            ' IP number or servername
    Database_Name = "hr"         ' Name of database
    User_ID = "root"                     ' id user or username
    Password = ""                    ' Password
    Table = "employees"                   ' Name of table to write to

    SQLStr = "SELECT * FROM " & Table
```

```
    Set Cn = New ADODB.Connection
    Cn.Open "Driver={MySQL ODBC 5.1 Driver};Server=" & Server_Name &
";Database=" & Database_Name & _
    ";Uid=" & User_ID & ";Pwd=" & Password & ";"

    rs.Open SQLStr, Cn, adOpenStatic

    With Worksheets(1).Cells ' Enter your sheet name and range here
      .ClearContents
      .CopyFromRecordset rs
    End With

    rs.Close
    Set rs = Nothing
    Cn.Close
    Set Cn = Nothing

End Sub
```

22. BASE DE REGISTRES

Fonction pour lire la valeur d'une clé de la base de registre:

```
'reads the value for the registry key i_RegKey
'if the key cannot be found, the return value is ""
Function RegKeyRead(i_RegKey As String) As String
Dim myWS As Object

  On Error Resume Next
  'access Windows scripting
  Set myWS = CreateObject("WScript.Shell")
  'read key from registry
  RegKeyRead = myWS.RegRead(i_RegKey)
End Function
```

Fonction pour contrôler si une clé existe dans la base de registres:

```
'returns True if the registry key i_RegKey was found
'and False if not
Function RegKeyExists(i_RegKey As String) As Boolean
Dim myWS As Object

  On Error GoTo ErrorHandler
  'access Windows scripting
  Set myWS = CreateObject("WScript.Shell")
  'try to read the registry key
  myWS.RegRead i_RegKey
  'key was found
  RegKeyExists = True
  Exit Function

ErrorHandler:
  'key was not found
  RegKeyExists = False
End Function
```

Fonction pour créer et/ou changer la valeur d'un clé existante dans la base de registre:

```
'sets the registry key i_RegKey to the
'value i_Value with type i_Type
'if i_Type is omitted, the value will be saved as string
'if i_RegKey wasn't found, a new registry key will be created
Sub RegKeySave(i_RegKey As String, _
               i_Value As String, _
    Optional i_Type As String = "REG_SZ")
Dim myWS As Object

  'access Windows scripting
  Set myWS = CreateObject("WScript.Shell")
  'write registry key
  myWS.RegWrite i_RegKey, i_Value, i_Type

End Sub
```

Fonction pour supprimer une clé de la base de registres:

```
'deletes i_RegKey from the registry
'returns True if the deletion was successful,
'and False if not (the key couldn't be found)
Function RegKeyDelete(i_RegKey As String) As Boolean
Dim myWS As Object

  On Error GoTo ErrorHandler
  'access Windows scripting
  Set myWS = CreateObject("WScript.Shell")
  'delete registry key
  myWS.RegDelete i_RegKey
  'deletion was successful
  RegKeyDelete = True
  Exit Function

ErrorHandler:
  'deletion wasn't successful
  RegKeyDelete = False
End Function
```

Ces fonctions utilisent une clé de registre avec le chemin complet de la clé, ainsi i_RegKey doit toujours commencer avec une des valeurs suivantes:

- `HKCU` or `HKEY_CURRENT_USER`
- `HKLM` or `HKEY_LOCAL_MACHINE`
- `HKCR` or `HKEY_CLASSES_ROOT`
- `HKEY_USERS`
- `HKEY_CURRENT_CONFIG`

et se terminer avec le nom de clé...

La fonction RegKeySave a un paramètre d'entrée qui définit le typage de la clé. Les valeurs supportées pour ce paramètre sont:

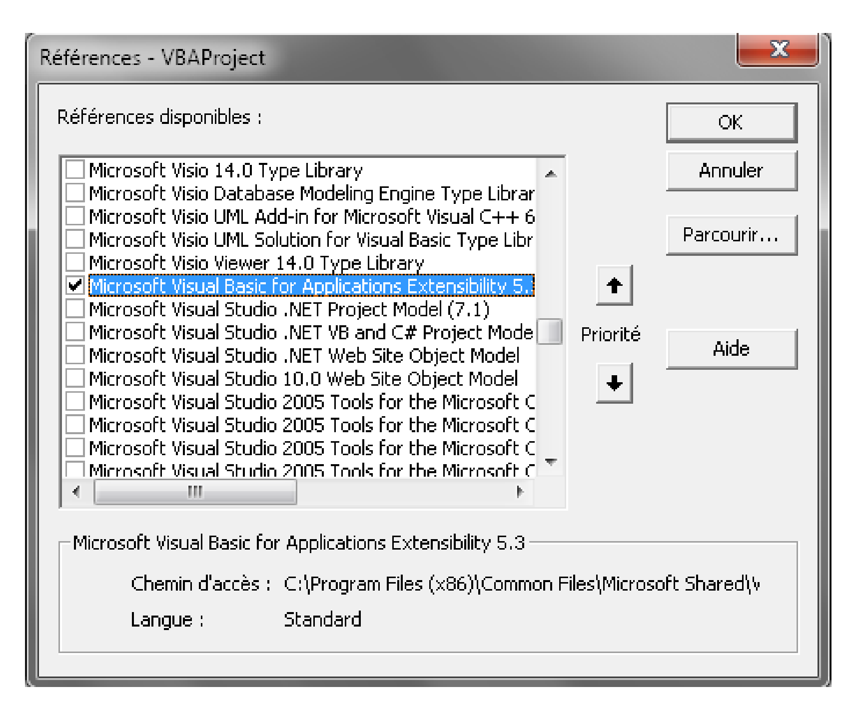
- `REG_SZ`: chaîne de caractères
- `REG_DWORD`. un nombre codés sur 32 bits
- `REG_EXPAND_SZ`: une chaîne de caractères étendue
- `REG_BINARY` - Binary data in any form. You really shouldn't touch such entries.

23. VBE

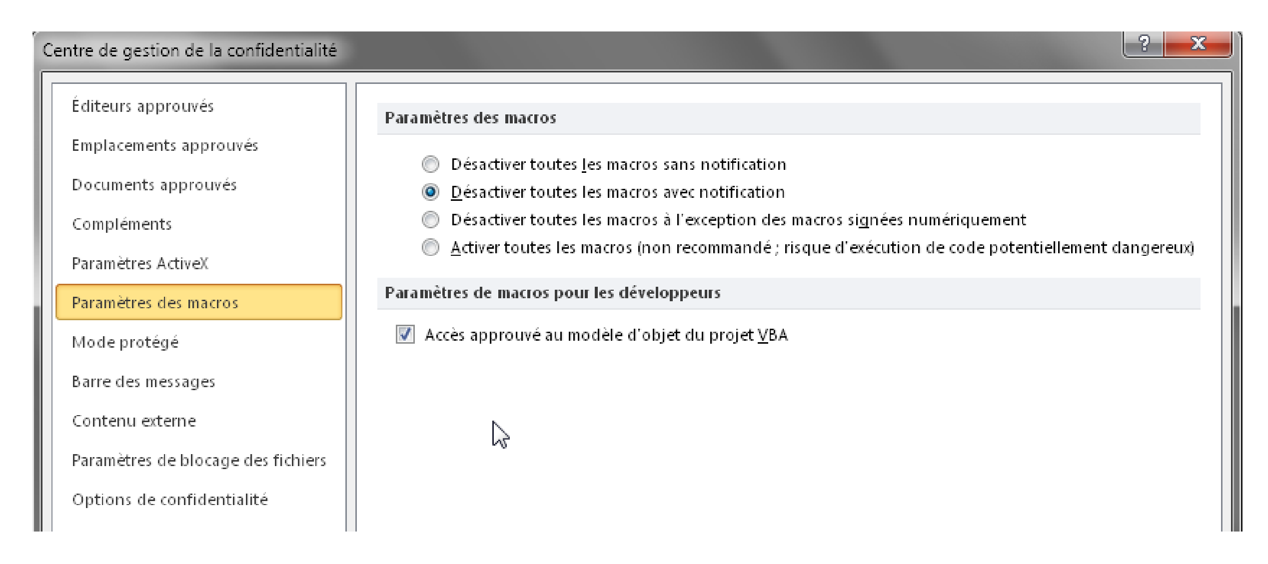
Il est possible de contrôler V.B.A. avec du V.B.A. Il est d'usage d'appeler cela le "VBE" car l'idée est de contrôle l'éditeur Visual Basic avec du V.B.A. C'est un domaine plutôt réservé aux consultants développeurs qui créent des applications complexes en entreprise aussi bien avec MS Excel qu'avec les autres outils de la suite MS Office (en particulier avec MS Access).

Il existe déjà un excellent PDF à ce sujet sur le web. Le but ici va seulement être de compléter celui-ci avec d'autres codes très utiles (qui proviennent aussi du web).

D'abord n'oubliez pas d'ajouter la référence V.B.E. (Visual Basic for Application Extensibility) au fichier:



Ensuite, dans Excel 2007 et 2010 il faudra autoriser l'accès du V.B.A. au V.B.A. en cochant **Accès approuvé au modèle d'objet du projet VBA**:



```
Sub OuvrirEditeurVBA()
   'Si vous faites de la gestion des erreurs il peut être utile d'ouvrir l'éditeur V.B.A. pour l'utilisateur connaisseur du code
   With Application.VBE.MainWindow
       .SetFocus
       .Visible = True
   End With
End Sub


'******************************
'Créateur(s): John WALKENBACH
'Dernière modification: 15.10.2013
'Nom procédure: AddButtonAndCode()
'Appelée par: -
'Commentaires: exemple de code V.B.A. qui génère un bouton ActiveX et y associe un code VBA
'******************************

Sub AddButtonAndCode()

   Dim NewSheet As Worksheet
   Dim NewButton As OLEObject

   'Nous ajoutons une feuille pour y mettre le bouton
   Set NewSheet = Sheets.Add
   'Nous ajoutons un bouton ActiveX sur la page
   Set NewButton = NewSheet.OLEObjects.Add("Forms.CommandButton.1")
   'Un peu d'esthétique
   With NewButton
       .Left = 4
       .Top = 4
       .Width = 100
       .Height = 24
       .Object.Caption = "Return to Sheet1"
   End With

   'Nous préparons le chaîne de code V.B.A. qui sera associée au bouton
   Code = "Sub CommandButton1_Click()" & vbCrLf
   Code = Code & "On Error Resume Next" & vbCrLf
   Code = Code & "Sheets(""Sheet1"").Activate" & vbCrLf
   Code = Code & "If Err <> 0 Then" & vbCrLf
   Code = Code & "MsgBox ""Cannot activate Sheet1.""" & vbCrLf
   Code = Code & "End If" & vbCrLf
   Code = Code & "End Sub"

   'Et nous associons le code
   With ActiveWorkbook.VBProject.VBComponents(NewSheet.Name).CodeModule
       NextLine = .CountOfLines + 1
       .InsertLines NextLine, Code
   End With

End Sub
```

```
'********************************
'Créateur(s): John WALKENBACH
'Dernière modification: 15.10.2013
'Nom procédure: ReplaceModule
'Appelée par: -
'Commentaires: Ensemble de routines pour mettre à jour/remplacer un module
V.B.A. par un autre
'********************************

Sub ReplaceModule()

   Dim ModuleFile As String
   Dim VBP As VBIDE.VBProject

   Set VBP = ActiveWorkbook.VBProject

   On Error GoTo ErrHandle

   With VBP.VBComponents
      'On supprime l'ancien module en supposant qu'il s'appelle Module2
      .Remove VBP.VBComponents("Module2")
      'On importe le nouveau module qui se trouve quelque part sur le
       disque. On pourrait au besoin l'exporter au prélable en tant que
       fichier texte avec
      '.Export "C:\Module2.txt"
      'On pourrait demander à l'utilisateur où est stockée la mise à jour
      mais on simplifie l'exemple en prenant en prenant un emplacement
      constant
      .Import "c:\" & "Update.bas"
      'le nom du module se trouve déjà dans le fichier *.bas
   End With

   MsgBox "Le module a été remplacé.", vbInformation
   Exit Sub

   ErrHandle:
      MsgBox "ERROR. The module may not have been replaced.",
vbCritical

End Sub
```

```
'*******************************
'Créateur(s): ?
'Dernière modification: 02.01.2014
'Nom procédure: DeleteAllVBACode
'Appelée par: -
'Commentaires: Routine qui supprime tous les modules et codes V.B.A. d'un
fichier
'*******************************

Sub DeleteAllVBACode()

   Dim VBProj As VBIDE.VBProject
   Dim VBComp As VBIDE.VBComponent
   Dim CodeMod As VBIDE.CodeModule

   Set VBProj = ActiveWorkbook.VBProject

   For Each VBComp In VBProj.VBComponents
      If VBComp.Type = vbext_ct_Document Then
          Set CodeMod = VBComp.CodeModule
          With CodeMod
            .DeleteLines 1, .CountOfLines
          End With
      Else
          VBProj.VBComponents.Remove VBComp
      End If
   Next VBComp

End Sub
```

```
'*******************************
'Créateur(s): ?
'Dernière modification: 02.01.2014
'Nom procédure: ListModules
'Appelée par: -
'Commentaires: Routine qui tous les modules et leurs types dans une feuille
Excel (en réalité le but est de savoir parcourir les modules)
'*******************************

Sub ListModules()

    Dim VBProj As VBIDE.VBProject
    Dim VBComp As VBIDE.VBComponent
    Dim WS As Worksheet
    Dim Rng As Range

    Set VBProj = ActiveWorkbook.VBProject
    Set WS = ActiveWorkbook.Worksheets("Sheet1")
    Set Rng = WS.Range("A1")

    For Each VBComp In VBProj.VBComponents
        Rng(1, 1).Value = VBComp.Name
        Rng(1, 2).Value = ComponentTypeToString(VBComp.Type)
        Set Rng = Rng(2, 1)
    Next VBComp

End Sub

Function ComponentTypeToString(ComponentType As
VBIDE.vbext_ComponentType) As String

    Select Case ComponentType
        Case vbext_ct_ActiveXDesigner
            ComponentTypeToString = "ActiveX Designer"
        Case vbext_ct_ClassModule
            ComponentTypeToString = "Class Module"
        Case vbext_ct_Document
            ComponentTypeToString = "Document Module"
        Case vbext_ct_MSForm
            ComponentTypeToString = "UserForm"
        Case vbext_ct_StdModule
            ComponentTypeToString = "Code Module"
        Case Else
            ComponentTypeToString = "Unknown Type: " &
CStr(ComponentType)
    End Select

End Function
```

```
'*******************************
'Créateur(s): ?
'Dernière modification: 26.05.2014
'Nom fonction: ModExiste
'Appelée par: -
'Commentaires: Fonction qui renvoie si un module existe ou non
'*******************************

Function ModExists(name As String) As Boolean

    ModExists = False
    Dim pVBE As VBIDE.VBE
    Set pVBE = Application.VBE
    Dim l As Long
    For l = 1 To pVBE.VBProjects.Count
        Dim k As Long
        For k = 1 To pVBE.VBProjects(l).VBComponents.Count
            If pVBE.VBProjects(l).VBComponents(k).Type = vbext_ct_StdModule
Then
                Dim s As String
                s = UCase(pVBE.VBProjects.Item(l).VBComponents(k).name)
                If s = UCase(name) Then
                    ModExists = True
                    Exit Function
                End If
            End If
        Next k
    Next l
End Function
```

```vba
'*******************************
'Créateur(s): ?
'Dernière modification: 02.01.2014
'Nom procédure: ListProcedures
'Appelée par: -
'Commentaires: Routine qui liste toutes les procédures du Module1 dans une
feuille Excel (en réalité le but est de savoir parcourir les procédures)
avec le nombre de lignes de code total et le numéro de ligne pour chaque
procédure
'*******************************

Sub ListProcedures()

    Dim VBProj As VBIDE.VBProject
    Dim VBComp As VBIDE.VBComponent
    Dim CodeMod As VBIDE.CodeModule
    Dim LineNum As Long
    Dim NumLines As Long
    Dim WS As Worksheet
    Dim Rng As Range
    Dim ProcName As String
    Dim ProcKind As VBIDE.vbext_ProcKind
    Set VBProj = ActiveWorkbook.VBProject
    Set VBComp = VBProj.VBComponents("Module1")
    Set CodeMod = VBComp.CodeModule
    Set WS = ActiveWorkbook.Worksheets("Sheet1")
    Set Rng = WS.Range("A1")

    MsgBox "There is " & CodeMod.CountOfLines & " lines of code"

    With CodeMod
        LineNum = .CountOfDeclarationLines + 1
        ProcName = .ProcOfLine(LineNum, ProcKind)
        Do Until LineNum >= .CountOfLines
            Rng(1, 1).Value = ProcName
            Rng(1, 2).Value = ProcKindString(ProcKind)
            Rng(1, 3).Value = LineNum
            Set Rng = Rng(2, 1)
            LineNum = LineNum + .ProcCountLines(ProcName, ProcKind) + 1
            ProcName = .ProcOfLine(LineNum, ProcKind)
        Loop
    End With
End Sub

Function ProcKindString(ProcKind As VBIDE.vbext_ProcKind) As String

    Select Case ProcKind
        Case vbext_pk_Get
            ProcKindString = "Property Get"
        Case vbext_pk_Let
            ProcKindString = "Property Let"
        Case vbext_pk_Set
            ProcKindString = "Property Set"
        Case vbext_pk_Proc
            ProcKindString = "Sub Or Function"
        Case Else
            ProcKindString = "Unknown Type: " & CStr(ProcKind)
    End Select
```

```
End Function

'*******************************
'Créateur(s): ?
'Dernière modification: 2016-01-21
'Nom procédure: DeleteVBA()
'Appelée par: -
'Commentaires: Code qui supprime tous les codes et composants VBA d'un
projet en cours
'*******************************

Public Sub DeleteAllCode()

    On Error Resume Next

    Dim x  As Integer
    Dim Proceed As VbMsgBoxResult
    Dim Prompt As String
    Dim Title As String

    Prompt = "Are you certain that you want to delete all the VBA Code
from " & _
    ActiveProject.name & "?"
    Title = "Verify Procedure"

    Proceed = MsgBox(Prompt, vbYesNo + vbQuestion, Title)
    If Proceed = vbNo Then
        MsgBox "Procedure Canceled", vbInformation, "Procedure Aborted"
        Exit Sub
    End If

    On Error Resume Next
    With ActiveProject.VBProject
        For x = .VBComponents.Count To 1 Step -1
            .VBComponents.Remove .VBComponents(x)
        Next x
        For x = .VBComponents.Count To 1 Step -1
            .VBComponents(x).CodeModule.DeleteLines _
            1, .VBComponents(x).CodeModule.CountOfLines
        Next x
    End With
    On Error GoTo 0

End Sub
```

24. CHANGEMENTS V.B.A. MICROSOFT EXCEL 2007

File search ne fonctionne plus depuis Excel 2007. Il faut maintenant utiliser la procédure suivante pour chercher, ouvrir ou traiter les fichiers d'un dossier donné (fonctionne avec Excel 2003 et 2007!):

Ce premier exemple ne nécessite pas de référence (libraire tierce):

```
Sub RechercheFichier()

   Dim strChemin, strFichier As String

   strChemin = "C:\dossier\"
   strFichier = Dir(strChemin & "*.xls") 'ne permet pas de rechercher dans
   des sous-dossiers

   Do While strFichier <> ""
      msgbox strFichier 'afficher le nom du ficher
      msgbox FileDateTime(strFichier) 'afficher la date de modif du fichier
      en string
      msgbox FileLen(strFichier) 'afficher la taille du fichier en bytes
      strFichier = Dir
   Loop

End Sub
```

Ce deuxième exemple nécessite la référence Microsoft Script Runtime:


```
Public Sub RechercheFichier()
    'Ne pas oublier d'ajouter la référence Microsoft Scripting Runtime
reference!!!!!!!!!!!!!!!!!!!!!
    Dim obj_Explorer As New FileSystemObject
    Dim str_Folder As Object 'It's the path to our files
    Dim xls_File As Variant 'It's the files we will loop on

    Set obj_Explorer = CreateObject("Scripting.FileSystemObject")
    Set str_Folder = obj_Explorer.GetFolder("C:\tmp\")

    'We loop through that folder
    For Each xls_File In str_Folder.Files
        Debug.Print xls_File.Name 'we just check file names
    Next xls_File
End Sub
```

Au passage voici une routine pour changer le nom de tous les fichiers (*.jpg dans l'exemple particulier ci-dessous) d'un dossier (la code est subtil dans le sens où que pour passer au fichier suivant il faut à la fin de la boucle appeler Dir( ) à vide):

```
Sub RenommerFichier()

    Dim strNomsFichiers As String
    Dim intNumeroFichier As Integer

    intNumeroFichier = 0
    ChDir "C:\tmp\images"
    strNomsFichiers = Dir("C:\tmp\images\*.jpg")
```

```
    Do While strNomsFichiers <> ""
        If InStr(strNomsFichiers, "Pic") <> 0 Then Exit Sub
        FileCopy strNomsFichiers, "Pic" & Format(intNumeroFichier,
"000") & ".jpg"
        intNumeroFichier = intNumeroFichier + 1
        Kill strNomsFichiers
        strNomsFichiers = Dir()
    Loop

End Sub
```

25. CHANGEMENTS V.B.A. MICROSOFT EXCEL 2010

Depuis Excel 2010 on peut enfin donner simplement une description des arguments d'une fonction simplement. Voici un exempleL

```
Function EXTRACTELEMENT(Txt, n, Separator) As String
     EXTRACTELEMENT = Split(Application.Trim(Txt), Separator,n - 1)
End Function
```

Avec sa description en tant que macro qui devra se lancer par exemple à l'ouverture du classeur:

```
Sub DescribeFunctionExtractelement()
   Dim FuncName As String
   Dim FuncDesc As String
   Dim Category As String
   Dim ArgDesc(1 To 3) As String

   FuncName = "EXTRACTELEMENT"
   FuncDesc = "Returns the nth element of a string that uses a separator
character"
   Category = 7 'Text category
   ArgDesc(1) = "String that contains the elements"
   ArgDesc(2) = "Element number to return"
   ArgDesc(3) = "Single-character element separator"

   Application.MacroOptions _
      Macro:=FuncName, _
      Description:=FuncDesc, _
      Category:=Category, _
      ArgumentDescriptions:=ArgDesc
      'L'aide des arguments en ligne n'est plus disponible en xlsx. Il faut
faire Ctrl+Shift+A après avoir ouvert la paranthèse de la fonction pour
qu'il affiche les arguments...
End Sub
```


Avec les catégories:

Numéro de Catégorie	Nom
0	Toute (non spécifiée)
1	Finance
2	Date & Heure
3	Math & Trigo
4	Statistiques
5	Recherche & Références
6	Bases de données
7	Logique
8	Informations

26. CONCLUSION

Nous avons vu au travers de ce PDF les bases du VBA.

27. TABLE DES FIGURES

28. INDEX